Docker Containers and Optix Edge

Bennati Luca

2025-07-02

FactoryTalk by ROCKWELL AUTOMATION

ASEM A ROCKWELL AUTOMATION COMPANY

Agenda

Introduction on containers

Docker containers hands-on session

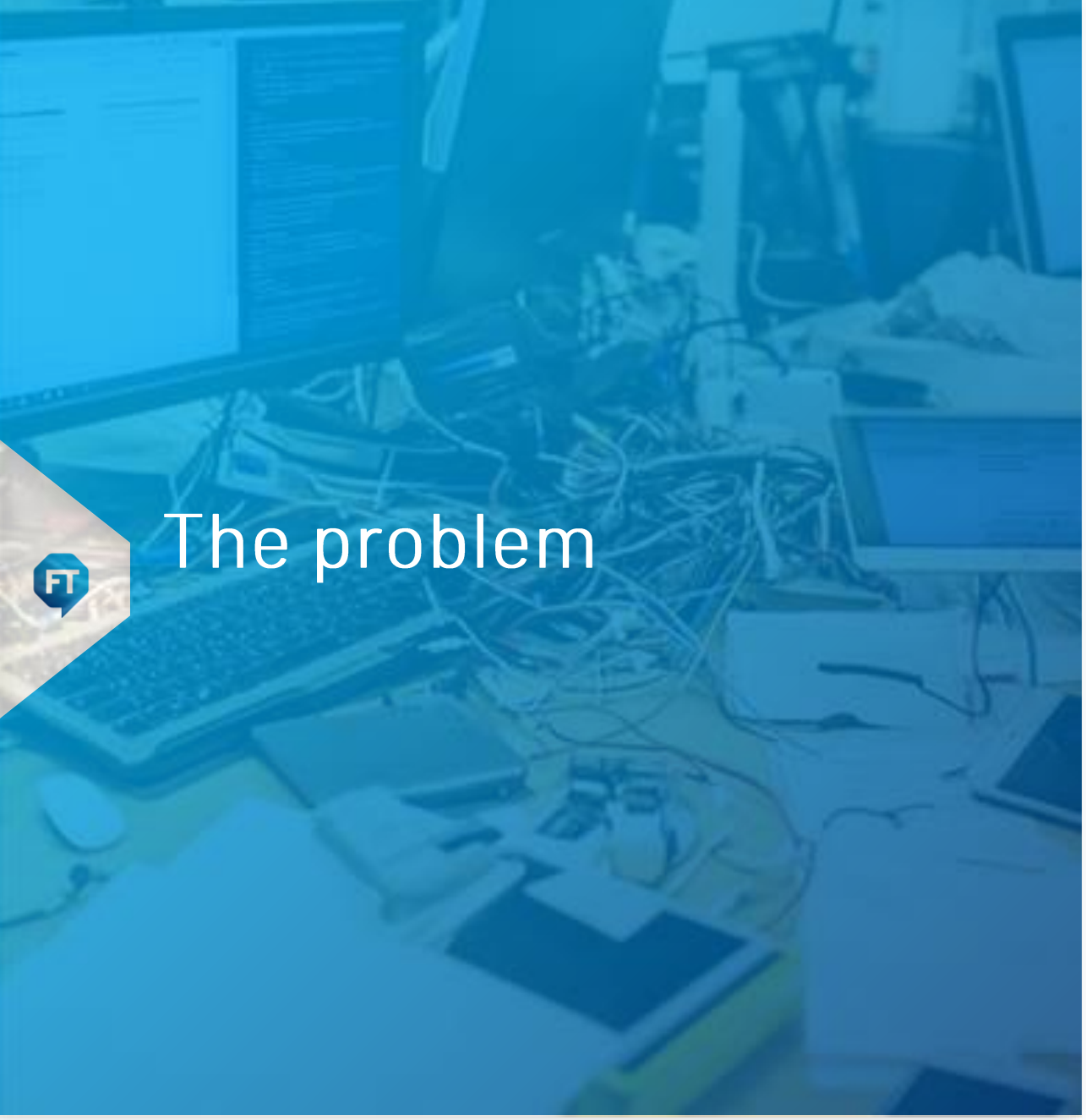
Docker containers on the Optix Edge

Introduction on containers

What are containers and why using them

But it works on my machine!

Every developer
Every single day



The problem

Your app runs perfectly on **your laptop**. But when deploying to **production**...

- Missing dependencies
- Drivers incompatibility
- Different O.S. versions
- Unexpected bugs
- Wrong CPU architecture
- Wrong app revision
- Unknown ZIP files called "MyApp_2025_V5_Final.zip"
- Windows updates

ASEM
A ROCKWELL AUTOMATION COMPANY

The solution

Packing all app resources into a box that can be deployed on the target

- Everything lives in the “box”:
  - Binaries
  - Dependencies

Packing techniques

Standard 10Kg trolley + 20Kg checked luggage… Not enough?

There are two type of person when preparing luggage for a vacation:

The overpacker

Vs.

The minimalist traveller

The overpacker: Virtual machines

Better bring the sofa, we might need it

A virtual machine is a completely **isolated environment** where everything lives in the VM itself

- The whole operating system is included
- Dedicated network connection can be used
- Provides complete isolation

Virtual machines

The pioneers of isolation

1964

IBM creates the M44/44X, an experimental system capable of emulating multiple IBM7044

1972

IBM releases the VM/370, the first operating system with native virtualization capabilities

1999

VMWare brings virtualization to desktop computers, revolutionizing development and testing

2006

AWS brings cloud virtualization to the masses with the EC2 platform

2010

Virtualization becomes a standard with new tools like KVM, VirtualBox, Hyper-V, Xen and many more

A ROCKWELL AUTOMATION COMPANY

Virtual machines

Why you should or should not use VMs

PROS	Strong isolation	OS-Level flexibility	High flexibility
CONS	Heavy	Slow startup	Resource intensive

“VMs are like standalone houses: secure, but heavy and slow to build. What if we could have something lighter, faster… but still isolated?”

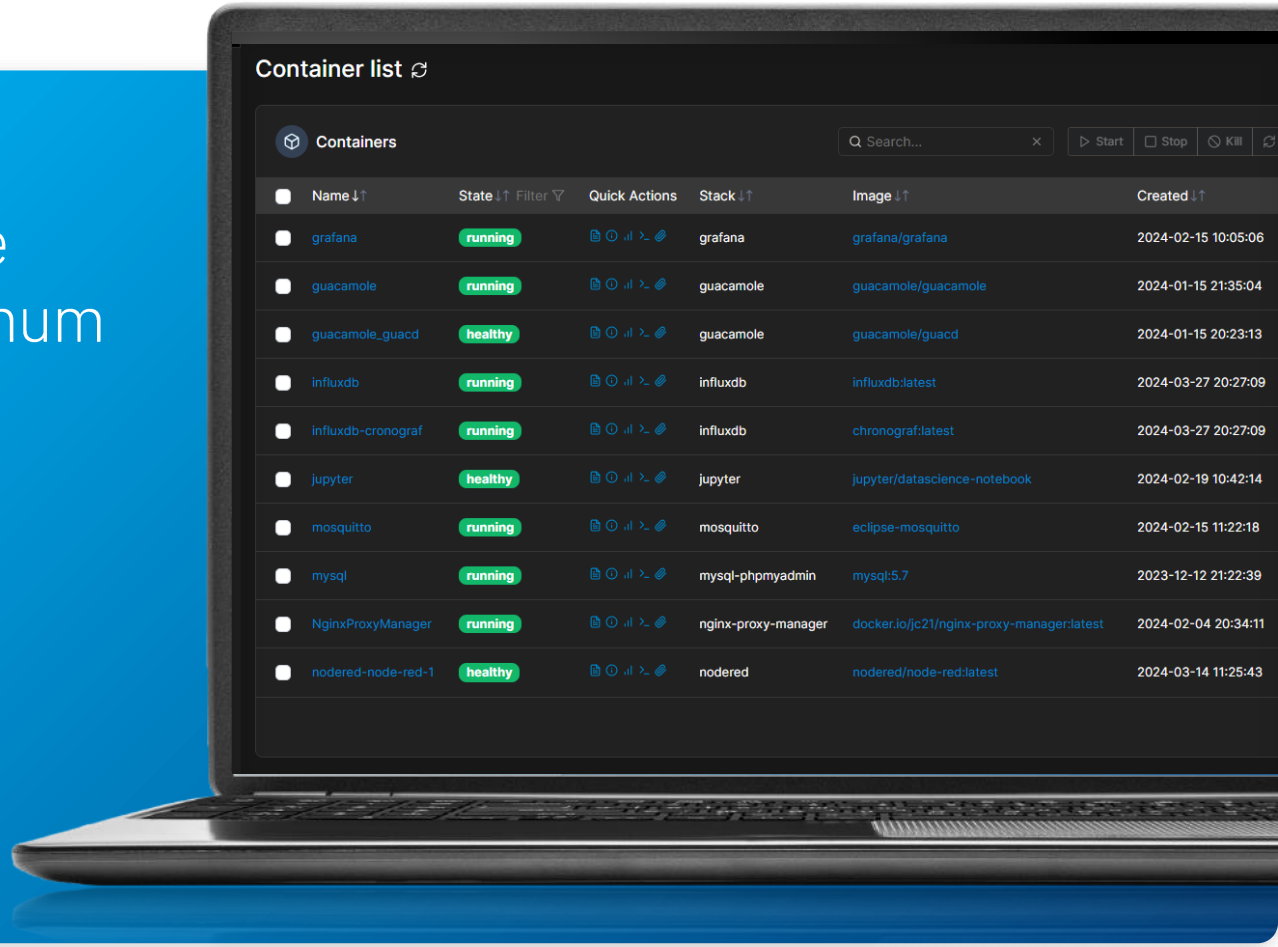
Microsoft Copilot
June 2025

The minimalist traveller: Containers

Do we really need shoes?

If there is no need to bring the whole O.S. we can just pack the bare minimum set of dependencies to run our app

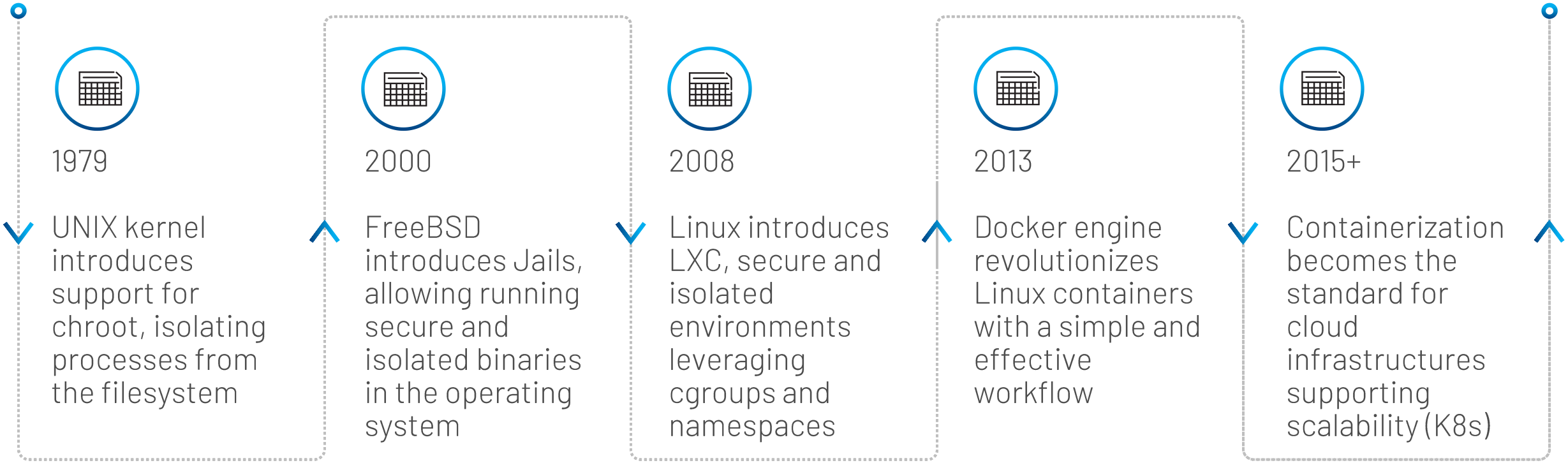
- Add the custom binaries
- Add the required libraries
- Pack it to a lightweight container



A ROCKWELL AUTOMATION COMPANY

Containers

A bare-minimum virtual machine



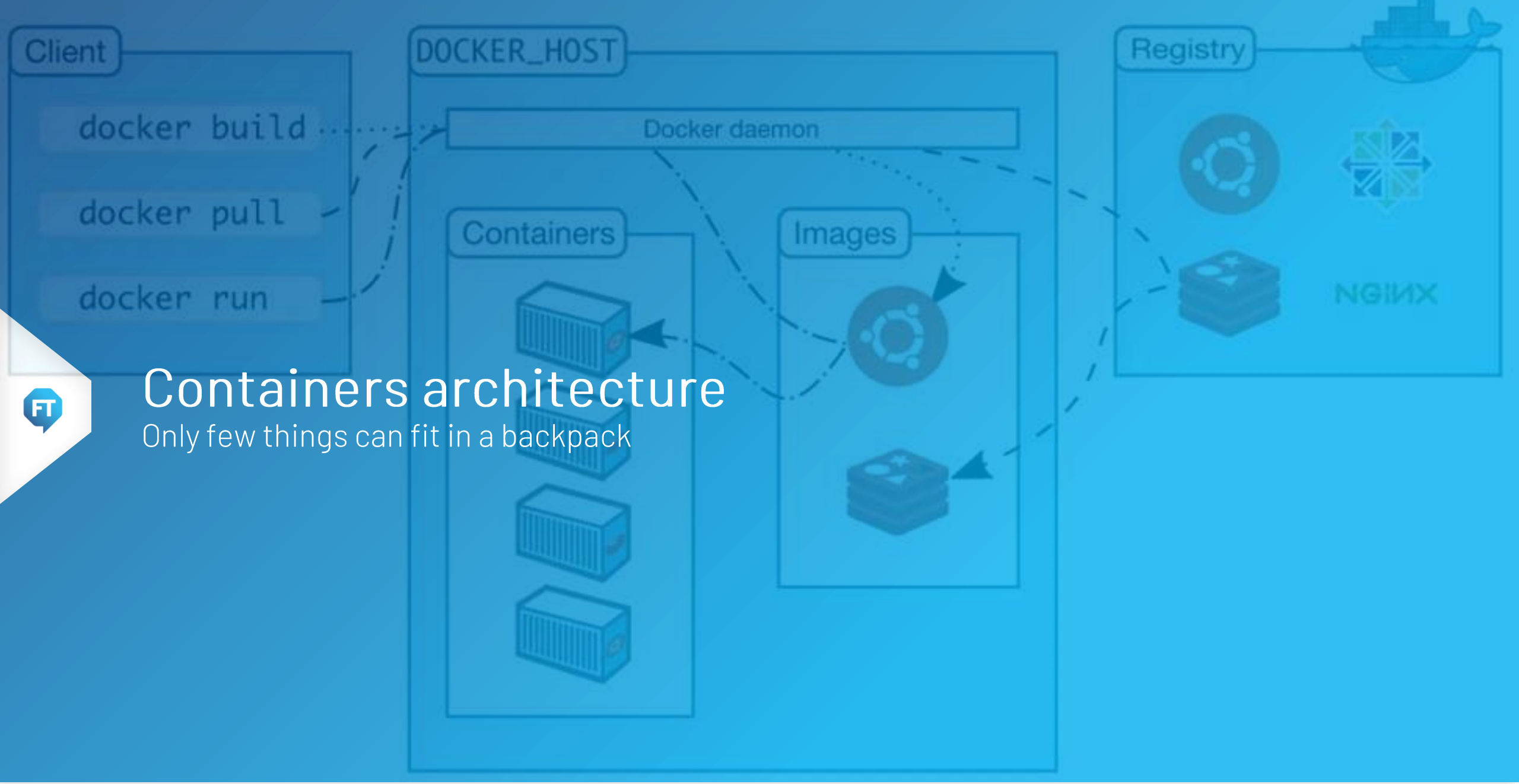
Containers

Less than a VM, more than an app

PROS	Light	Fast startup	Portable	Scalable
CONS	Weaker isolation	More complex[1]	Volatile storage[2]	OS dependency with some I/O[3]

1. Some manual scripting knowledges are needed. Together with Linux operating system knowledges
2. Volumes can be bound to specific folders to make portions of the container persistent
3. Only when working with phisical I/O interfaces or other low-level resources

ASEM
A ROCKWELL AUTOMATION COMPANY

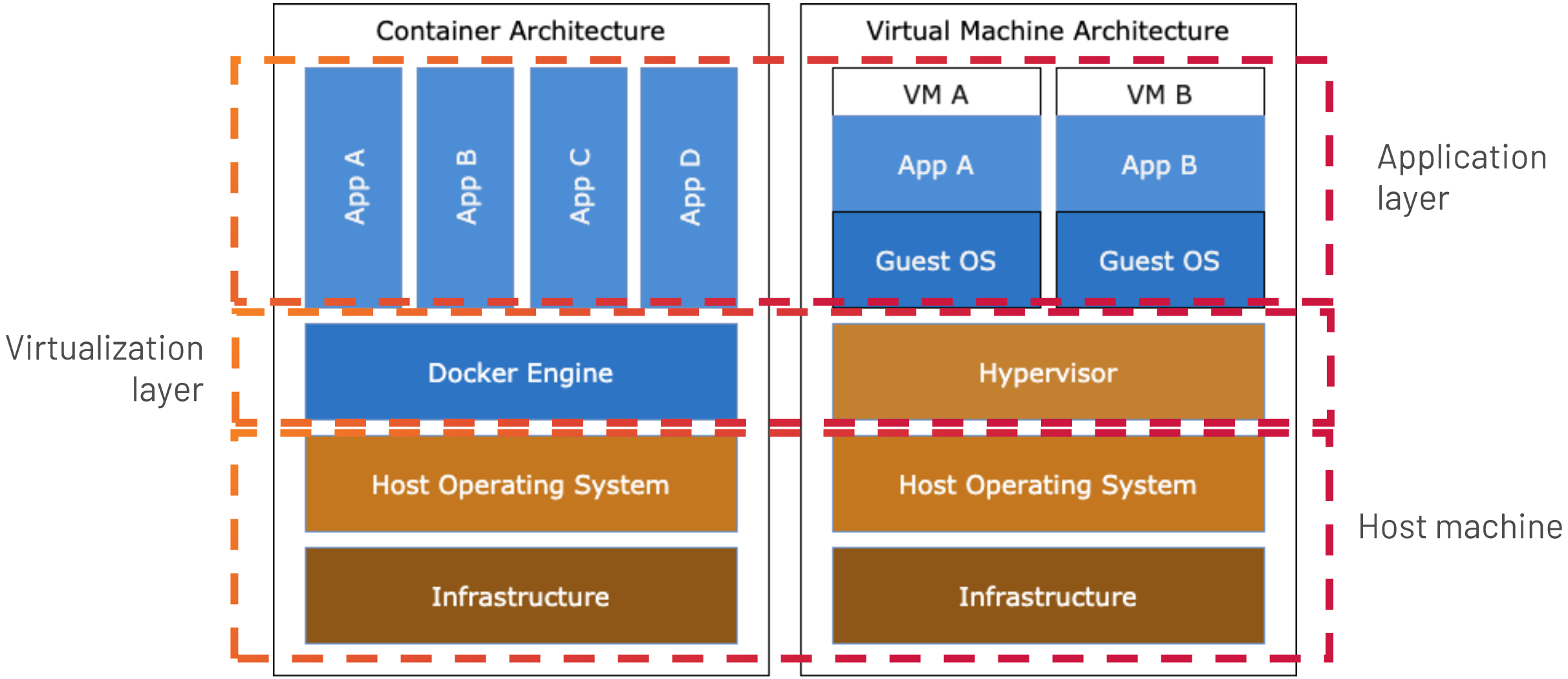


Containers architecture

Only few things can fit in a backpack

Containers vs. Virtual Machines

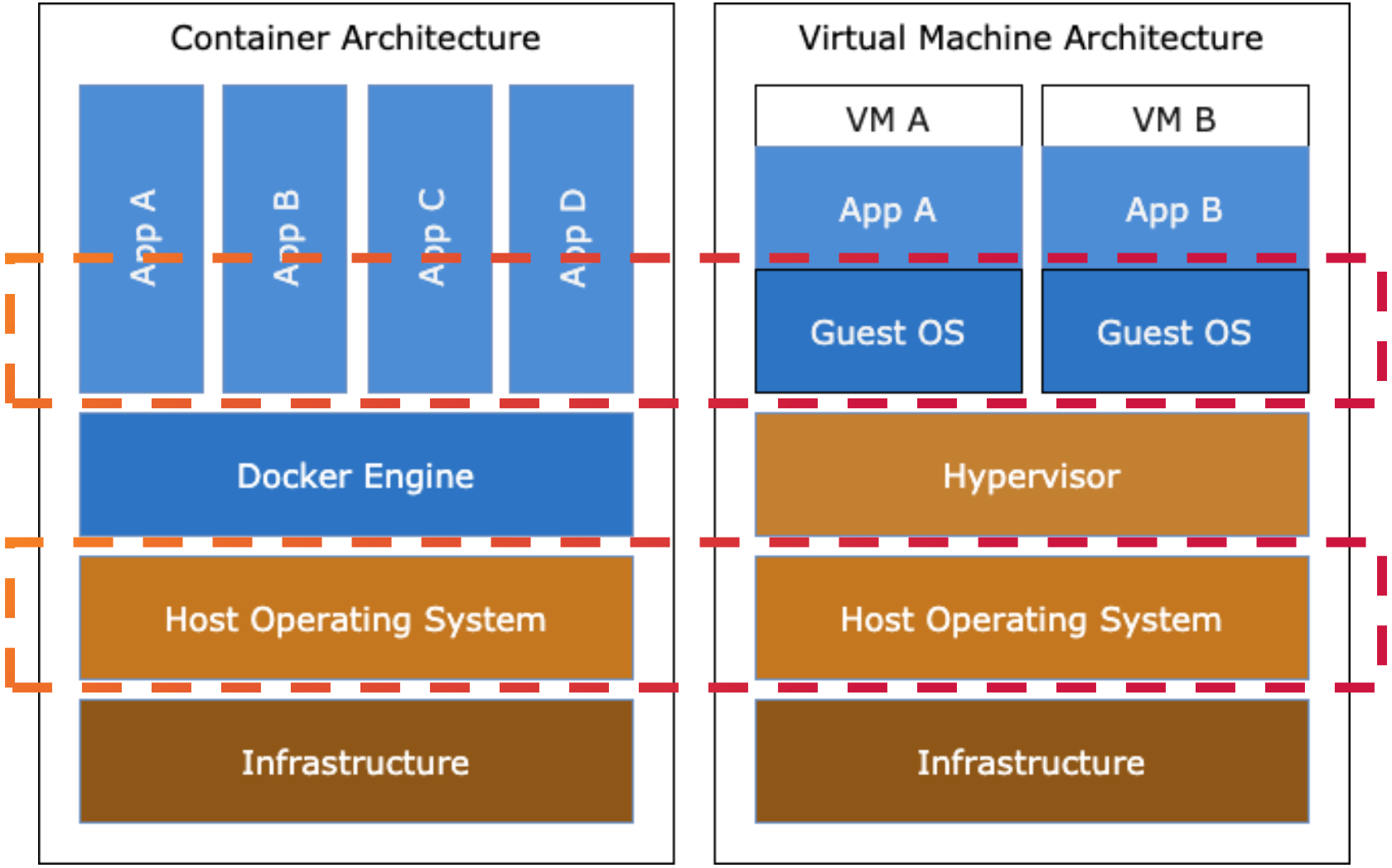
So close, yet so far



A lot of unnecessary duplicates

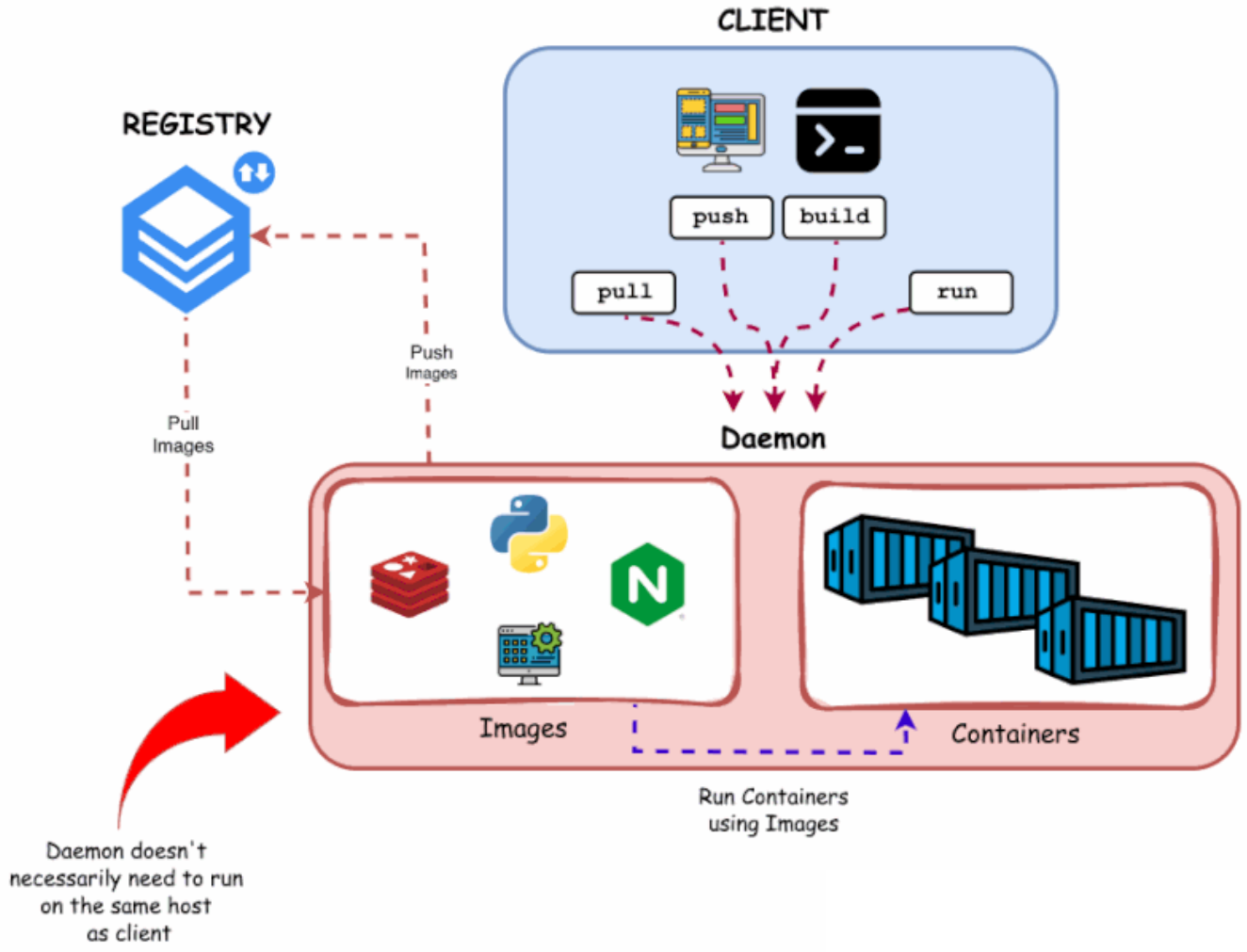
Remove everything unneeded

An O.S. is already in the machine, no need to waste resources



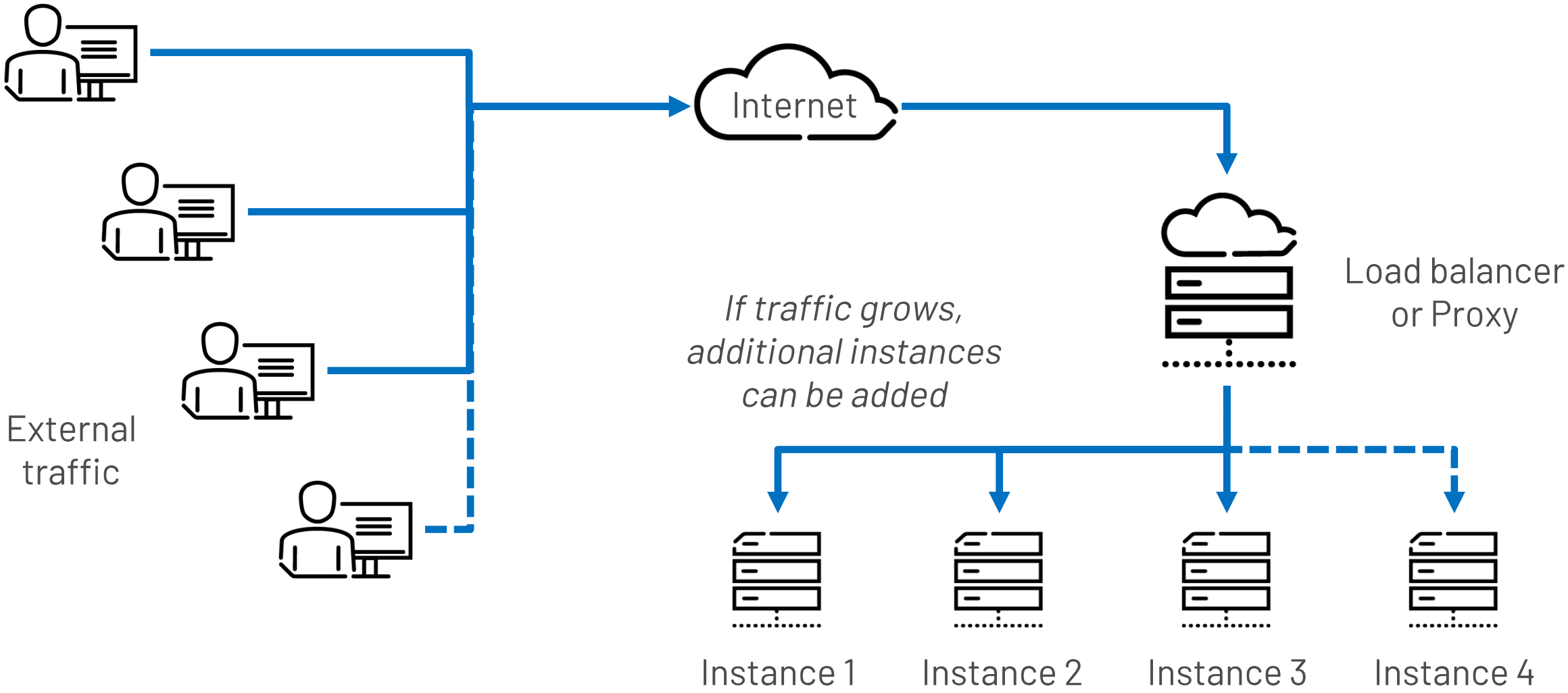
Containers vs. images

You can't have a container without an image



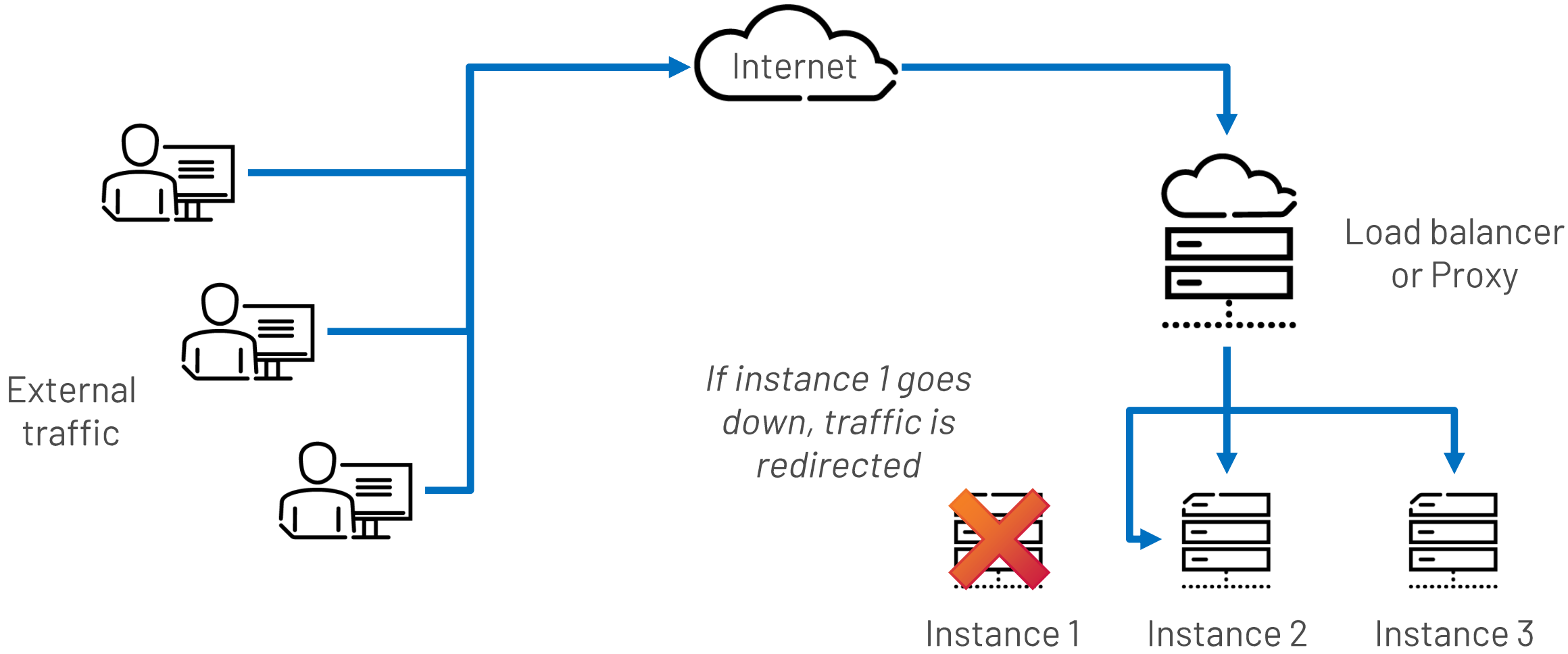
Redundancy and scalability

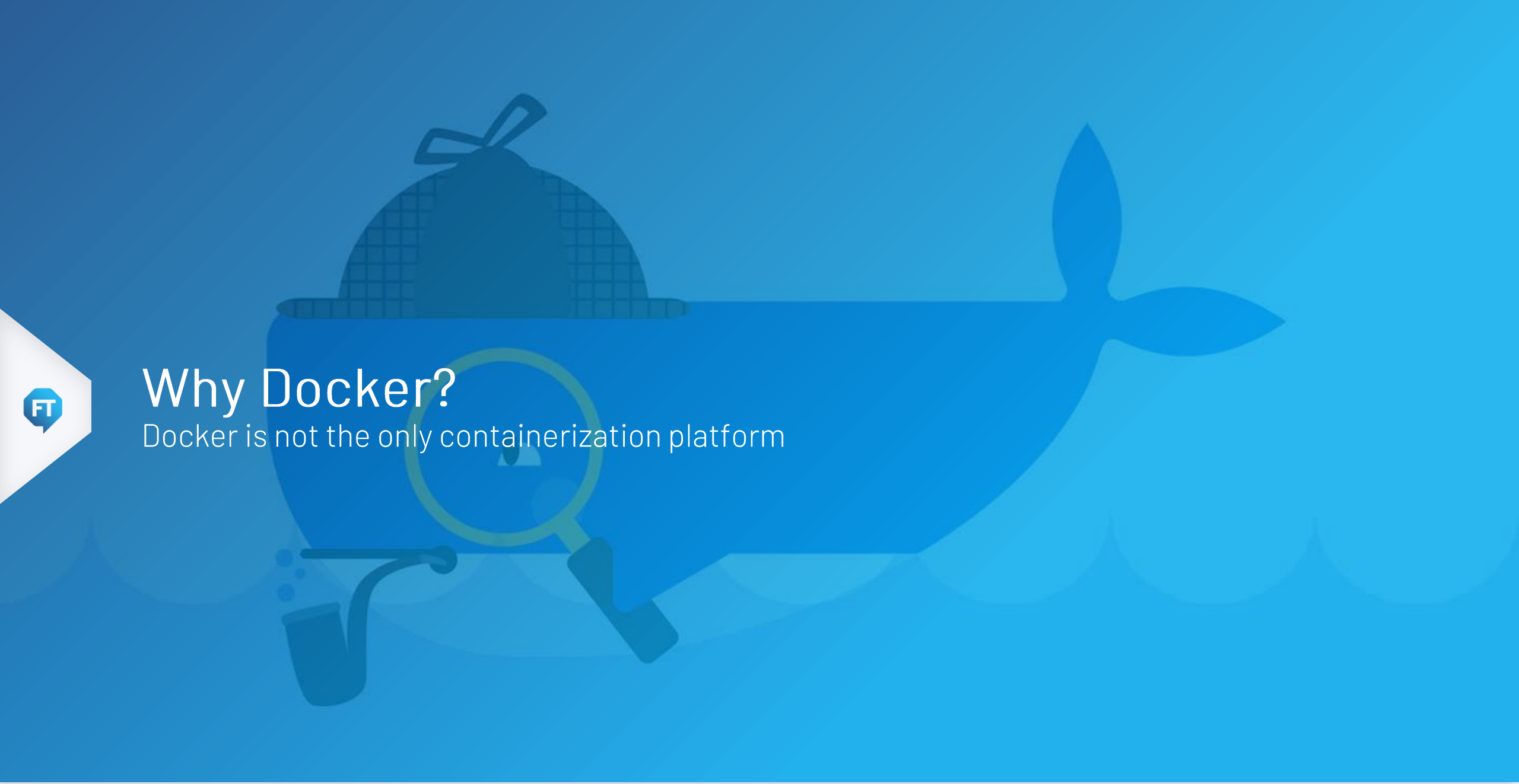
One container, two containers, three containers, ...



Redundancy and scalability

One container, two containers, three containers, ...





Why Docker?

Docker is not the only containerization platform

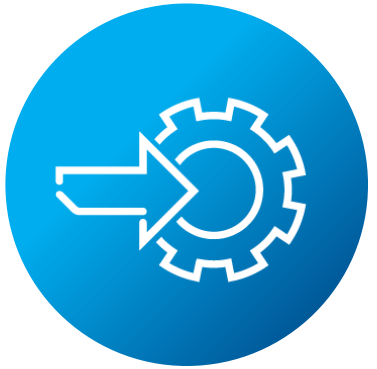
Why Docker?

Docker is not the only containerization platform

- Simple CLI
- Intuitive commands
- Good documentation

- Standardized images format
- Wide support across different devices

- Wide variety of pre-built containers



- Easy to integrate with CI/CD pipelines

- Customizable with plugins
- Support for clusters

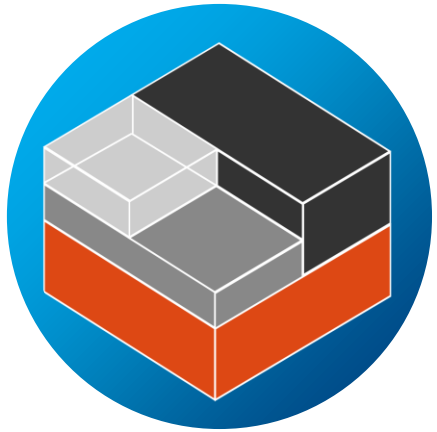
ASEM
A ROCKWELL AUTOMATION COMPANY

Other containerization platforms

Every job requires the right tools

LXC

More low-level, less user-friendly



Rkt (CoreOS)

Was focused on enterprise usage but now deprecated

Podman

Gaining traction but still not famous as Docker

Containerd

Being now hidden behind most containerization softwares, nobody uses it “raw”

OpenVZ

Requires kernel patching and other advanced intervention on the O.S.

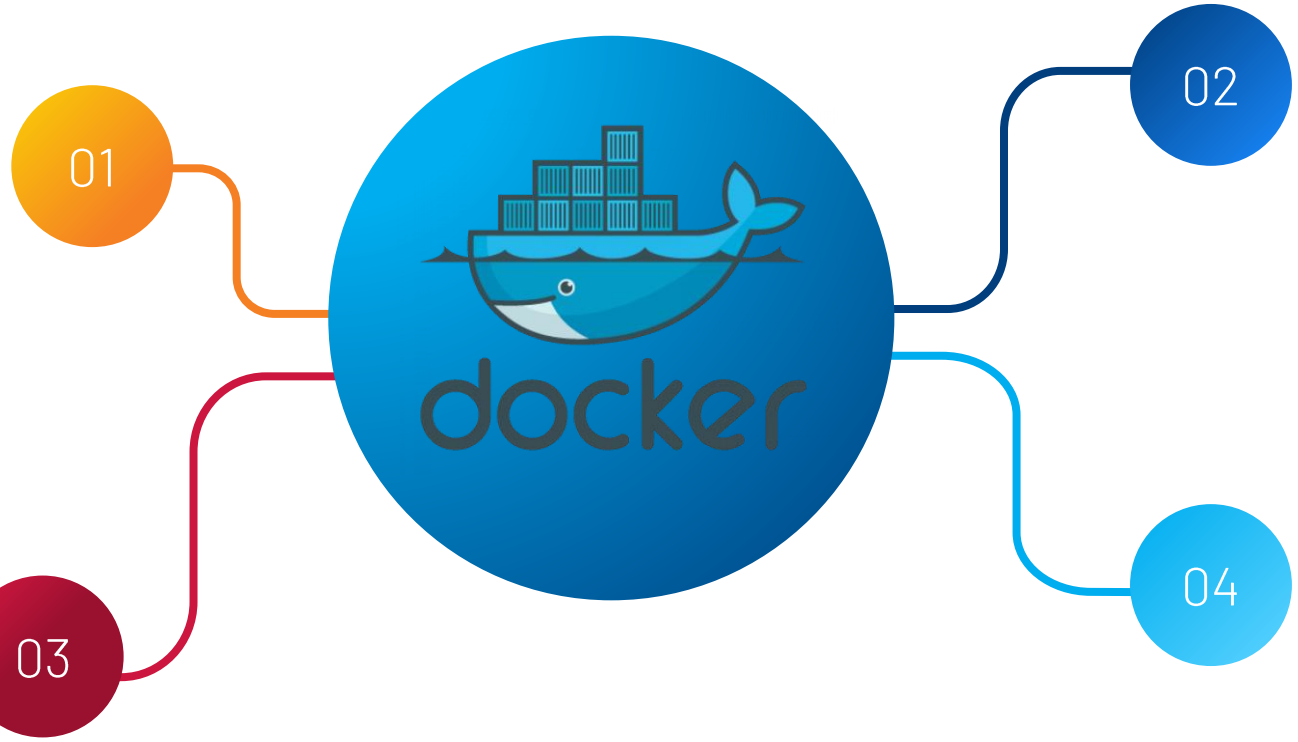
Closer to a VM than to a container

Creating a simple image

What are the building blocks of an image?

Image building blocks

To create an image, we only need a few things



Base image

Contains the minimum set of libraries to complete a task

Requirements and dependencies

We can add custom requirements or components if needed

Custom binaries

The actual app we need to run

Entrypoint

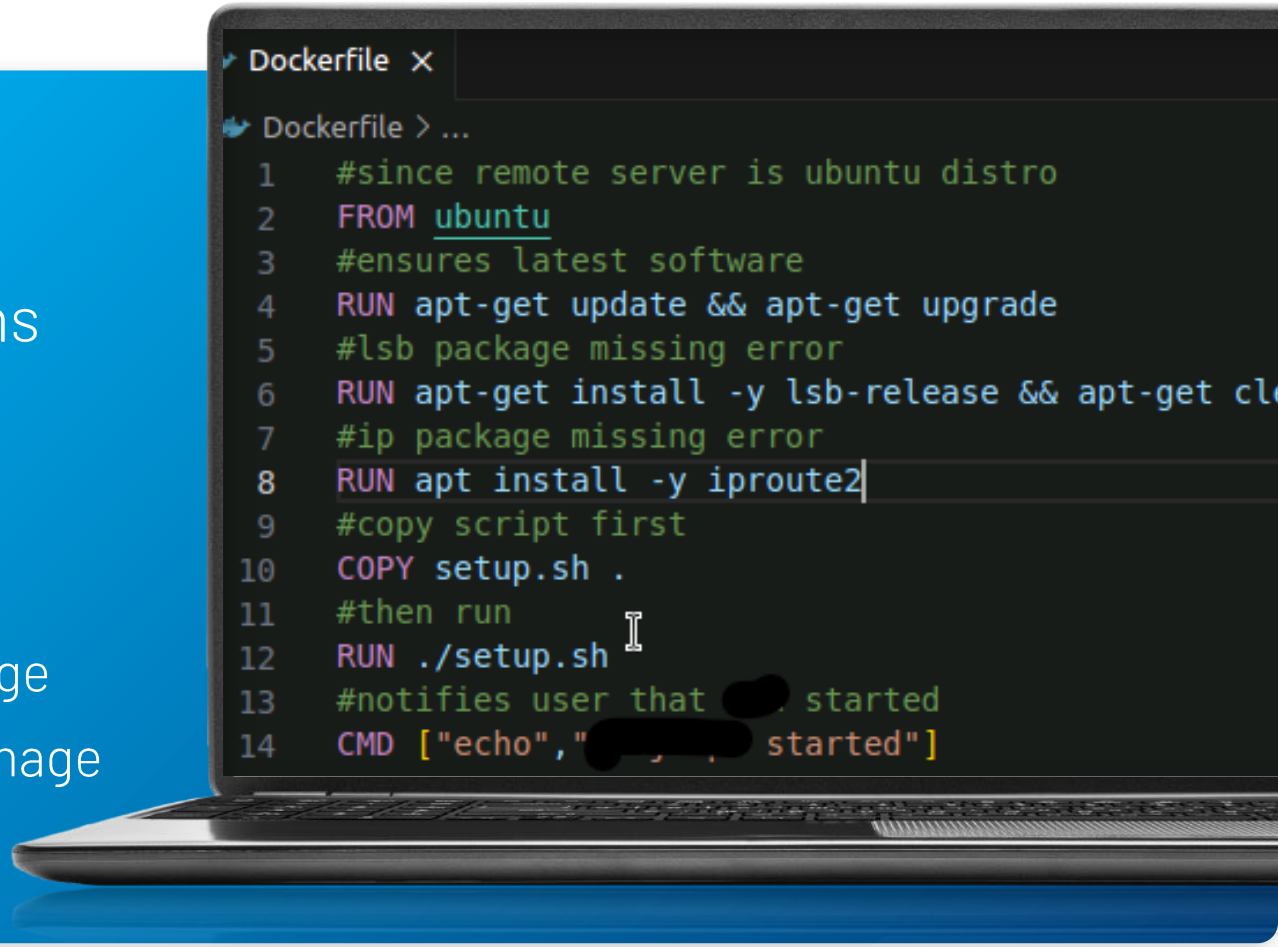
Tells the Docker engine what to do with the image

The Dockerfile

All we need is a text file

Docker containers are built from a Dockerfile where a set of instructions describes the build workflow

- FROM → Base image to start with
- RUN → Set of custom commands
- COPY → Copies the binaries to the image
- CMD → What to do when starting the image



ASEM
A ROCKWELL AUTOMATION COMPANY

Hello world!

The very first step for all developers

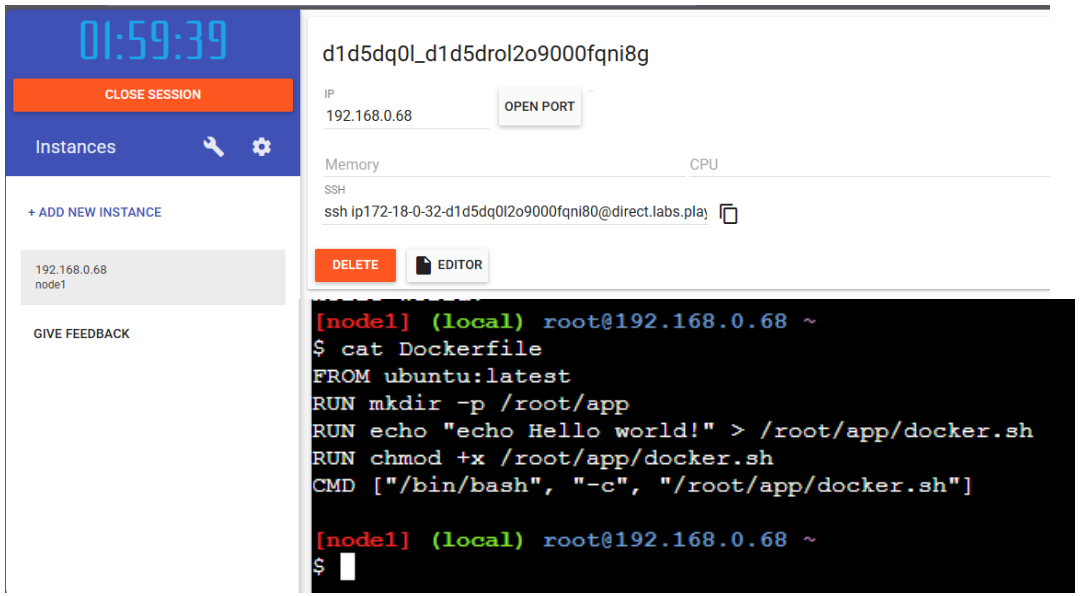
Hello world!

Bare minimum container image

- Login to https://labs.play-with-docker.com/
- Create a Dockerfile:
  - Type: `vi Dockerfile`
  - Press `i` to start insert mode
  - Type:

```
FROM ubuntu:latest
RUN mkdir -p /root/app
RUN echo “echo Hello World” > /root/app/docker.sh
RUN chmod +x /root/app/docker.sh
CMD [“/bin/bash”, “-c”, “/root/app/docker.sh”]
```

  - Press `esc` to stop edit mode
  - Type `:wq` to save and exit



Hello world!

Bare minimum container image

- Build the image with: `docker build -t hello-world .`
- Run the image with: `docker run hello-world`
  - An «Hello World» message should be printed

```
[node1] (local) root@192.168.0.68 ~
$ docker build -t hello-world .
[+] Building 0.5s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 213B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/ubuntu:latest@sha256:b59d21599a2b151e23eea5f6602f4af4d7d31c4
 => CACHED [2/4] RUN mkdir -p /root/app
 => CACHED [3/4] RUN echo "echo Hello world!" > /root/app/docker.sh
 => CACHED [4/4] RUN chmod +x /root/app/docker.sh
 => exporting to image
 => => exporting layers
 => => writing image sha256:3ce0bdd4c6ee1ff37ae76ed1e512206a901e66a270865955d2ab43d0a66dfe93
 => => naming to docker.io/library/hello-world
[node1] (local) root@192.168.0.68 ~
$ docker run hello-world
Hello world!
[node1] (local) root@192.168.0.68 ~
$
```

Customizing an existing image

Let's build a very simple web server

Hello world!

Same message, but now as a web page

- Login to https://labs.play-with-docker.com/
- Create a Dockerfile:
  - Type: `vi Dockerfile`
  - Press `i` to start insert mode
  - Type:

```
FROM nginx
RUN echo “Hello World!” > /usr/share/nginx/html/index.html
CMD  [“nginx-debug”, “-g”, “daemon off;”]
```

  - Press `esc` to stop edit mode
  - Type `:wq` to save and exit

Hello world!

Same message, but now as a web page

- Build the container with `docker build -t web-test .`
- Run the container with `docker run -it -p 8080:80 web-test`
- Wait for the container to start
- Check the web page using "Open Port" and type port 8080

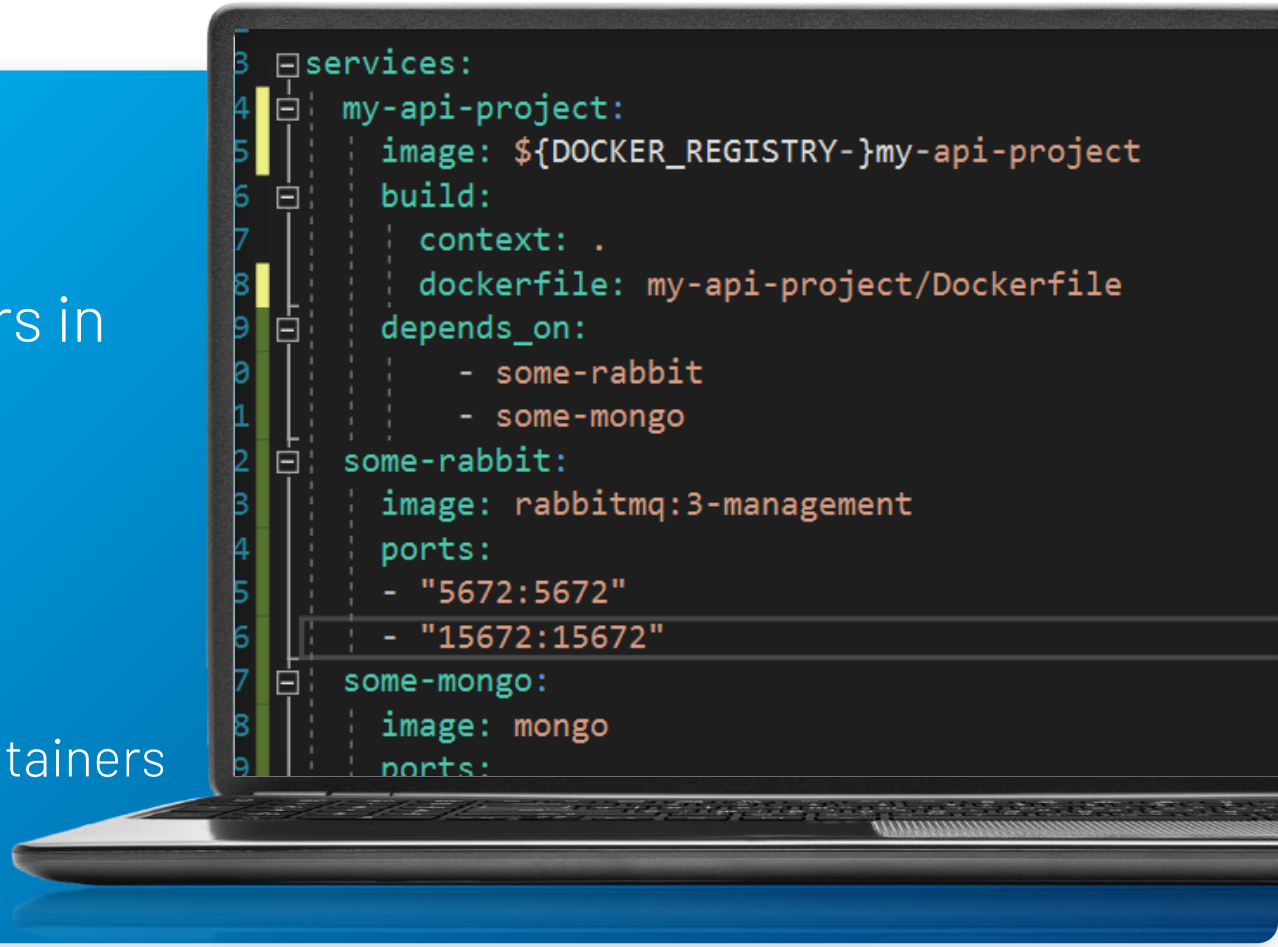
```
[node1] (local) root@192.168.0.68 ~
$ docker build -t web-test .
[+] Building 0.5s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 148B
 => [internal] load metadata for docker.io/library/nginx:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/nginx:latest@sha256:6784fb0834aa7
 => CACHED [2/2] RUN echo "Hello world!" > /usr/share/nginx/html/i
 => exporting to image
 => => exporting layers
 => => writing image sha256:802a4461e1bc55429dc336f742cb9dec234dd3
 => => naming to docker.io/library/web-test
[node1] (local) root@192.168.0.68 ~
$ docker run -it -p 8080:80 web-test
```

The Composer plugin

Composer = Docker x container^2

Composer is a Docker plugin to configure and run multiple containers in a single shot

- Supports defining a startup order
- Allows configuration for all containers
- Allows loading parameters from env
- Creates a dedicated network for all containers

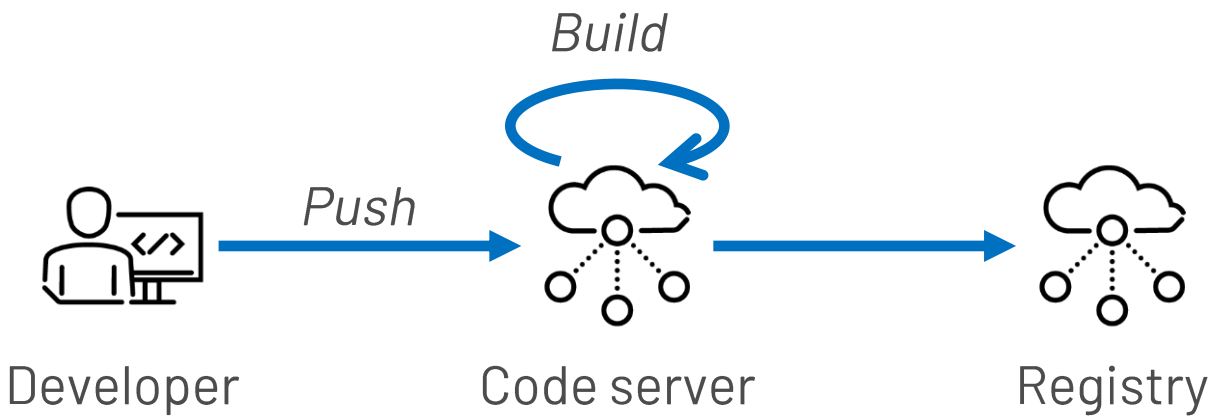


A ROCKWELL AUTOMATION COMPANY

So, why the Composer plugin is so used?

Modularity and flexibility makes your life easier

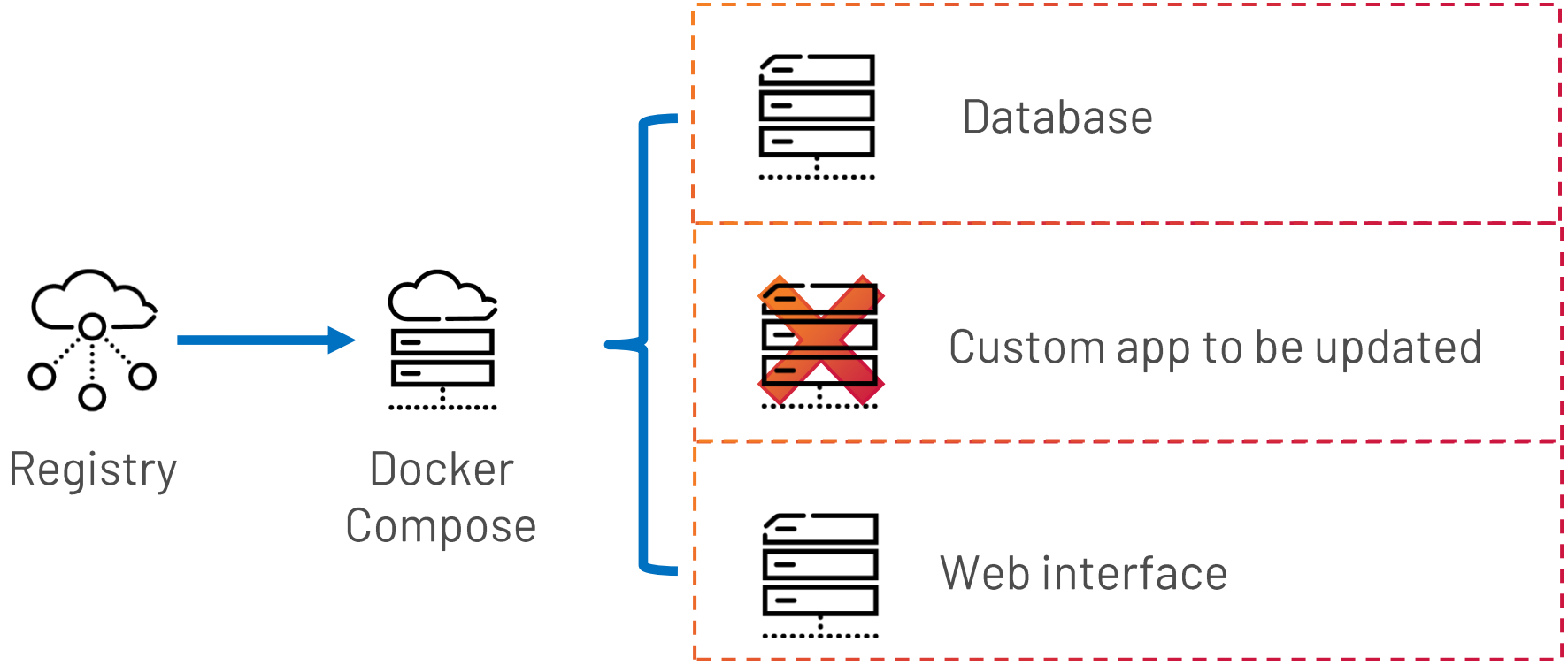
- Splitting the application into separate containers makes development easier and faster
  - Makes CI and CD workflow much simpler as each block can be built and tested independently



Patching a running environment

Let's leverage the flexibility of Docker compose and redundancy techniques

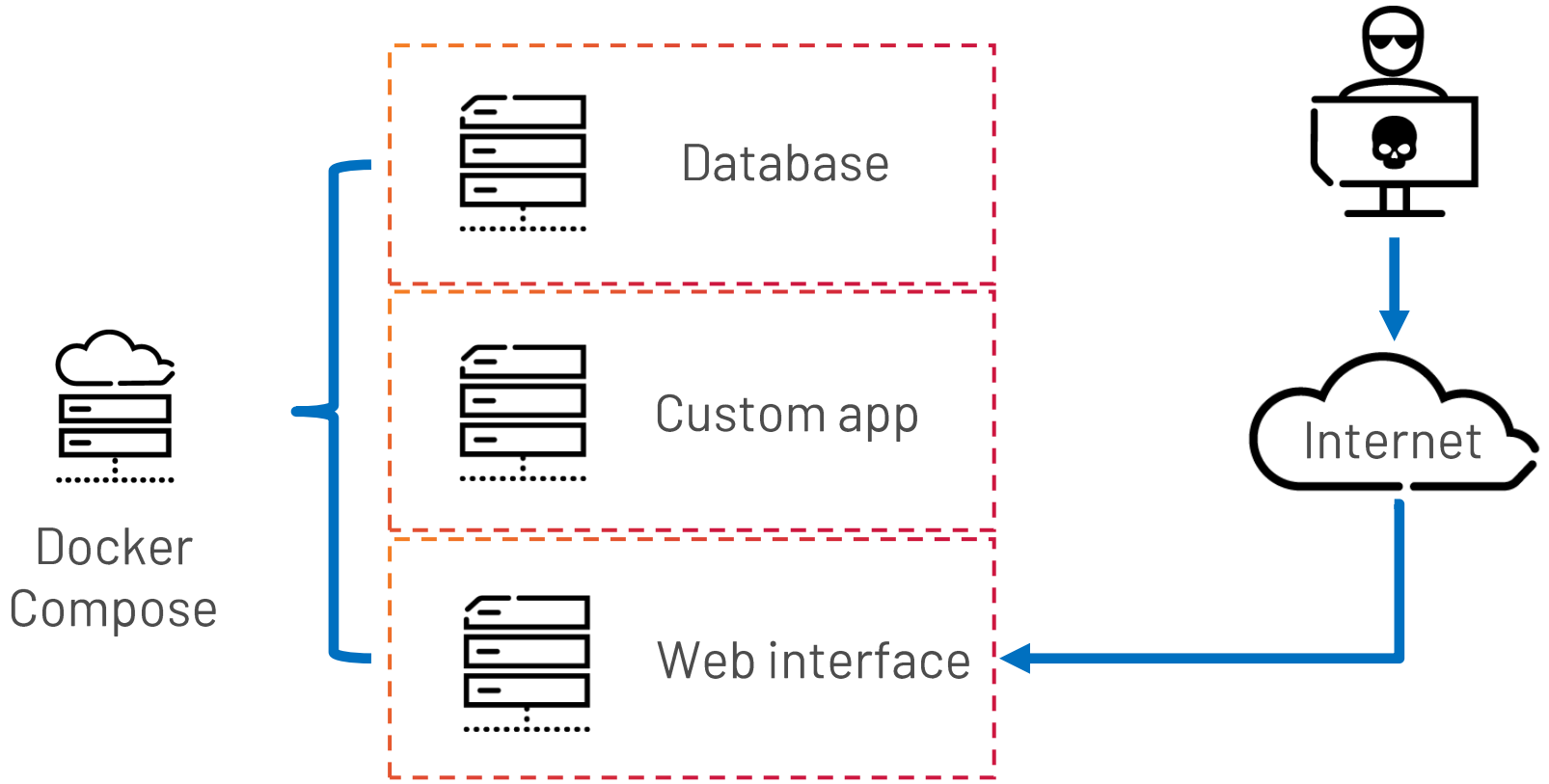
- If a block needs updates or maintenance, there's no need to rebuild the whole server
  - The container can be temporarily moved to a different machine
  - No need to edit other building blocks of the application



Reducing attack surface

Divide et impera

- Each container is an independent block
- Each container can live on a different machine



MySQL + PhpMyAdmin

A database engine with its cool user interface

MySQL + PhpMyAdmin

Copy-paste is welcome

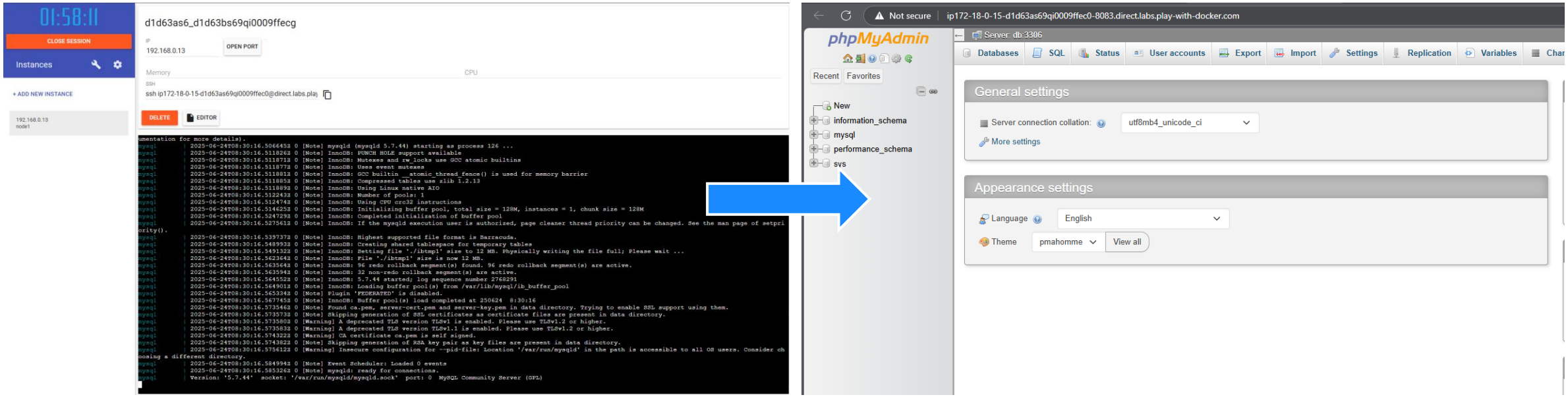
- Login to https://labs.play-with-docker.com/
- Create a Docker compose file:
  - Type: `vi docker-compose.yaml`
  - Press `i` to start insert mode
  - Paste the content on the right
  - Press `esc` to stop edit mode
  - Type `:wq` to save and exit

```
services:
  mysql:
    image: mysql:5.7
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: my_secret_password
    restart: unless-stopped
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadmin
    depends_on:
      - mysql
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
      PMA_USER: root
      PMA_PASSWORD: my_secret_password
    restart: unless-stopped
    ports:
      - 8083:80
```

MySQL + PhpMyAdmin

Can we test it?

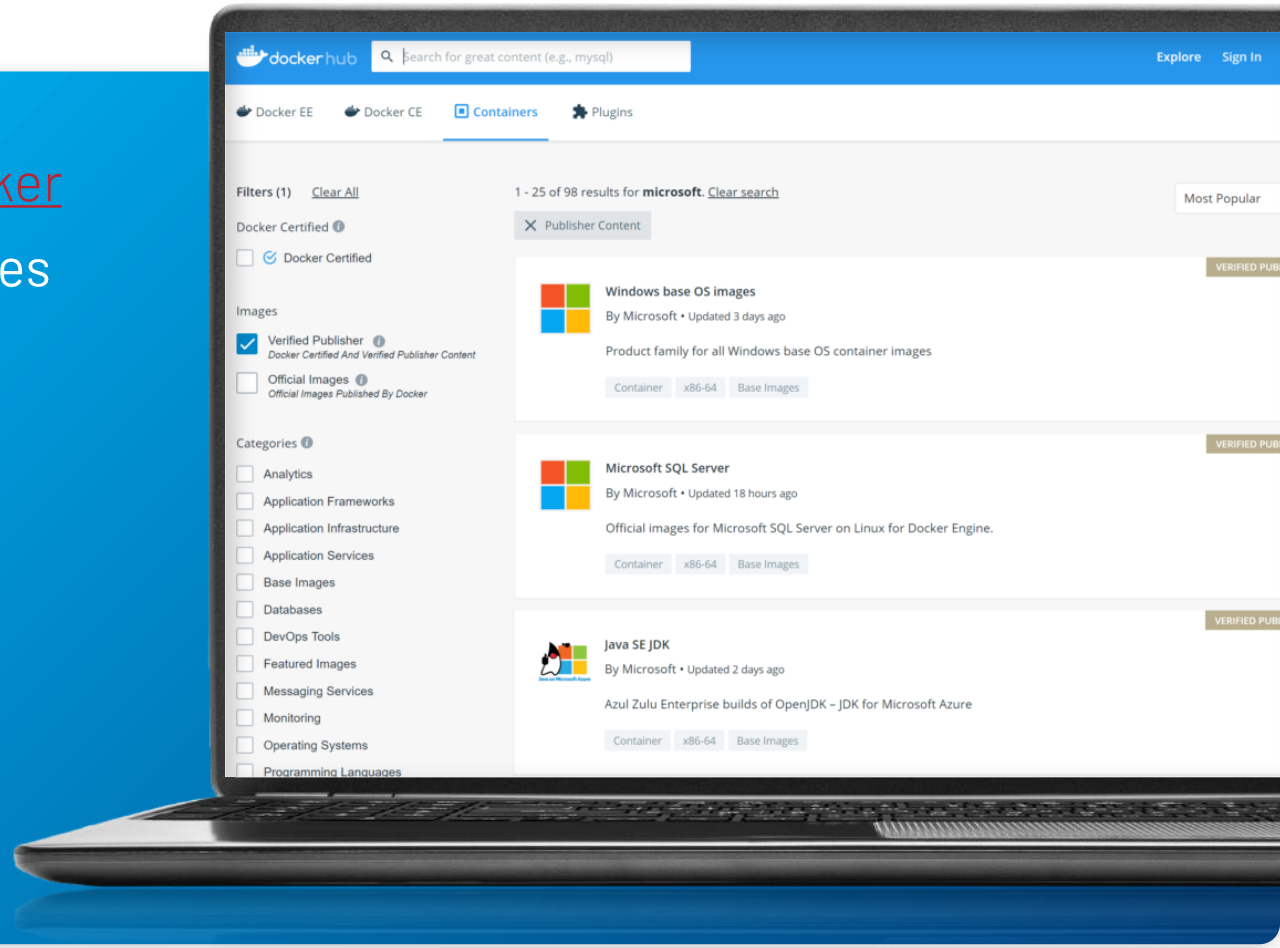
- Start the containers with `docker compose up`
- Click the «Open Port» button and select port 8083

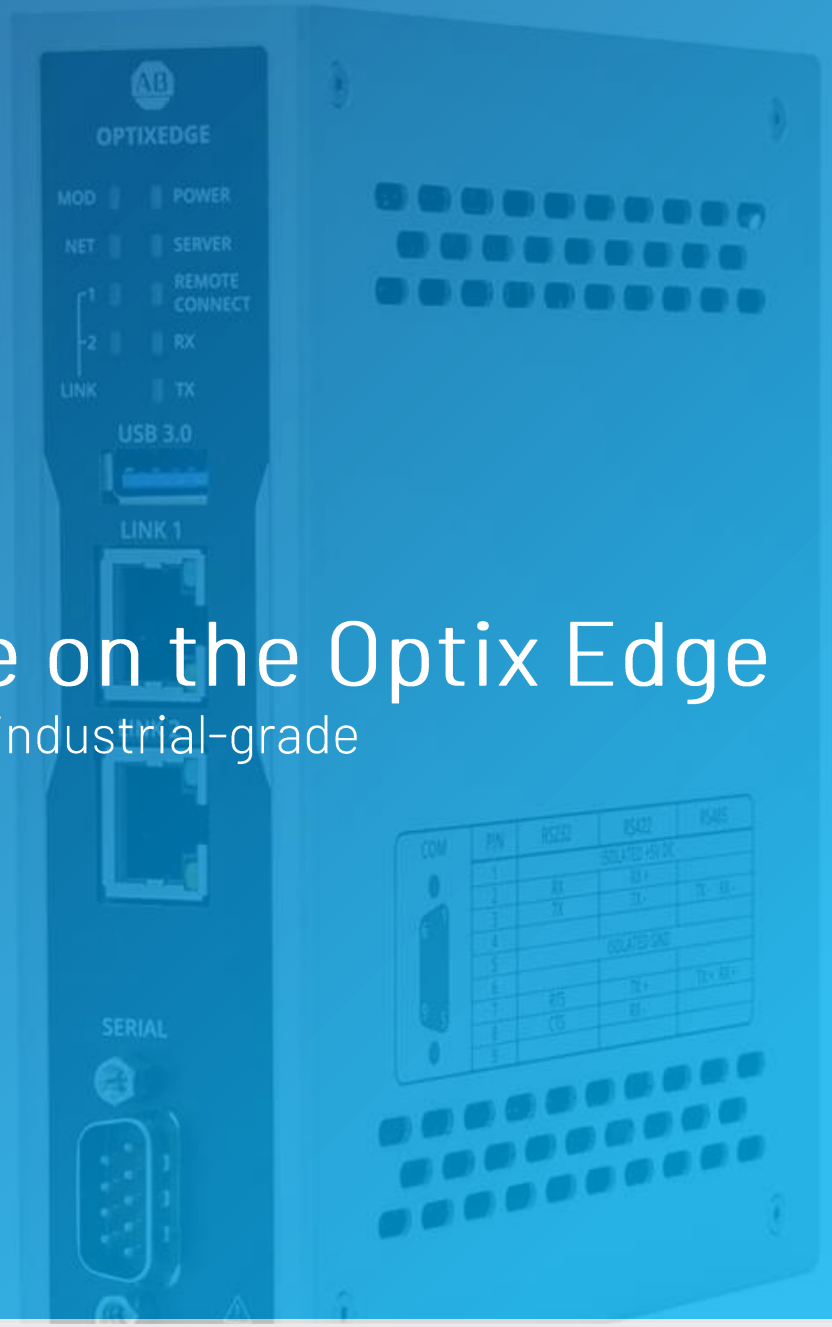


Want to learn more?

Once you start using Docker, you'll never stop

- hotheadhacker/awesome-selfhost-docker
  - Giant collection of useful Docker images
- Docker Hub
  - The main registry where to download images
- Docker Desktop | Docker Docs
  - Extensive Docker documentation





Docker engine on the Optix Edge

Let's make Docker more industrial-grade

ASEM
A ROCKWELL AUTOMATION COMPANY

Enabling the Docker engine

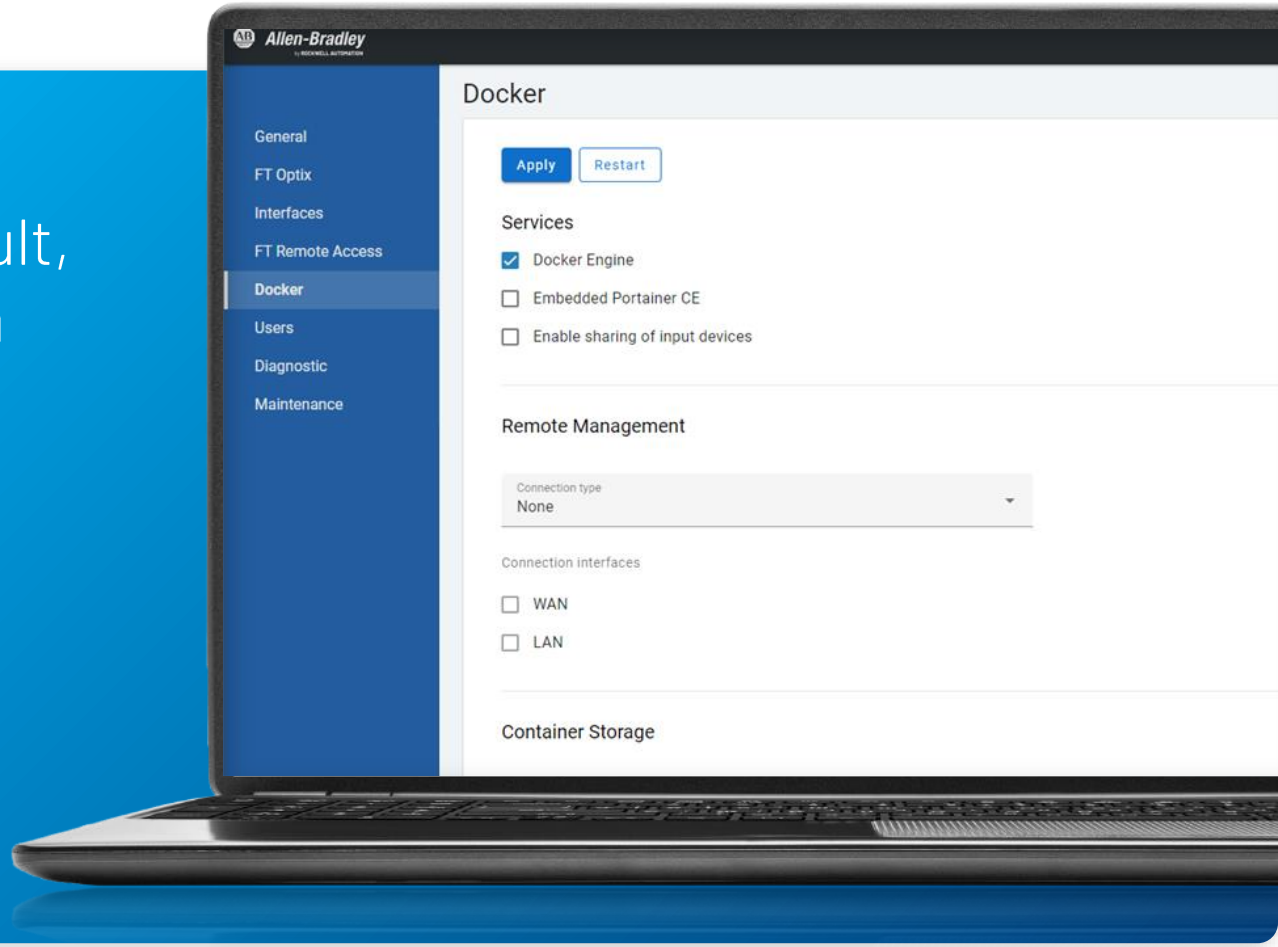
Tick, save, reboot

Enabling the Docker engine

All we need is a text file

Docker engine is not enable by default, user can activate it from the System Manager interface

- Tick the option «Docker engine»
- Reboot the device



Loading a Docker container with System Manager

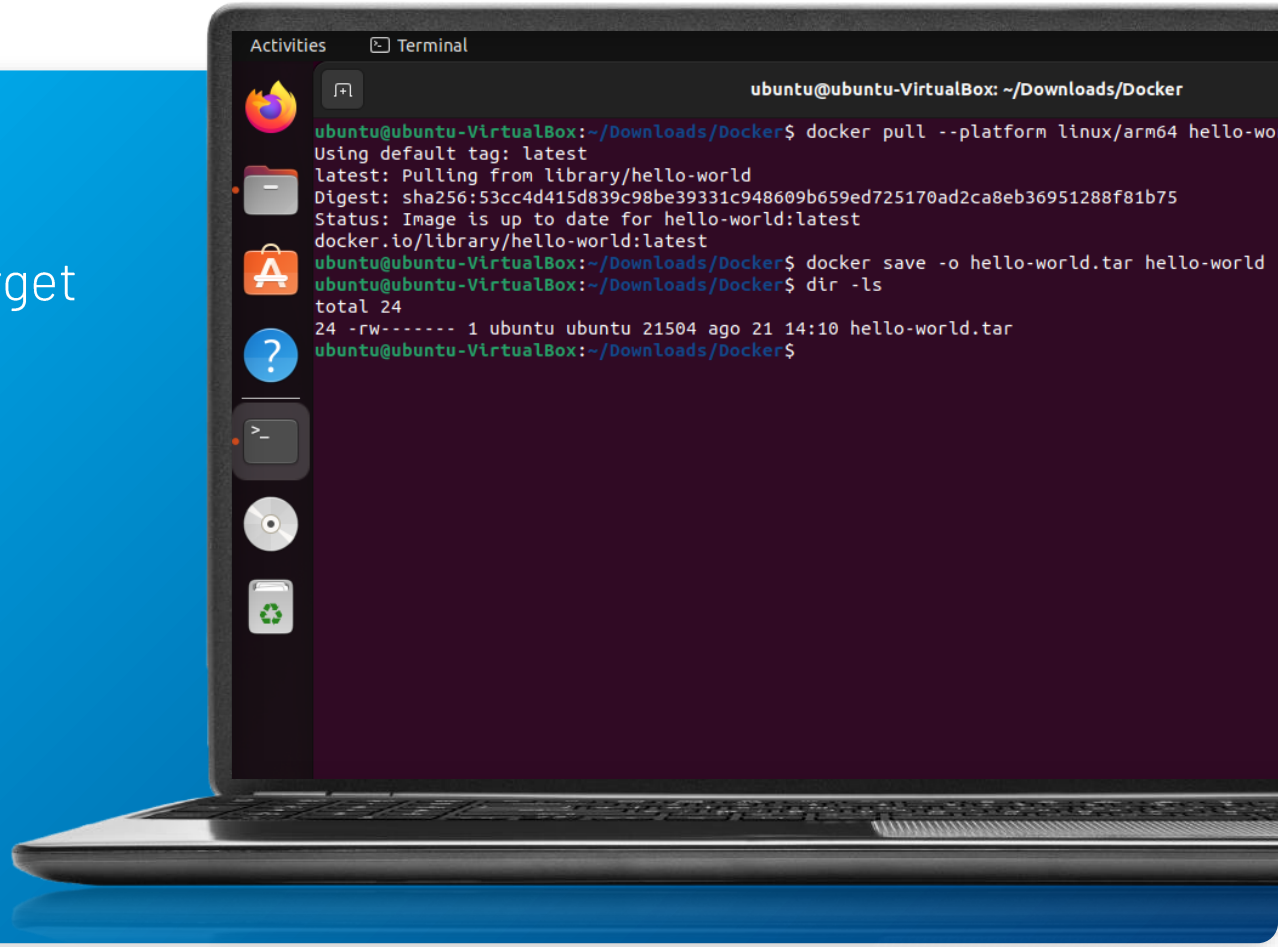
Warning: a USB key is needed

Loading containers from USB

Load a docker-compose.yaml and the relevant .tar image

- Pull the image from any registry
- Make sure to specify «linux/arm64» as target
- Export image to tar file
- Copy to USB
- Do not compress it!

```
$ docker pull --platform linux/arm64 hello-world
$ docker save -o hello-world.tar hello-world
```



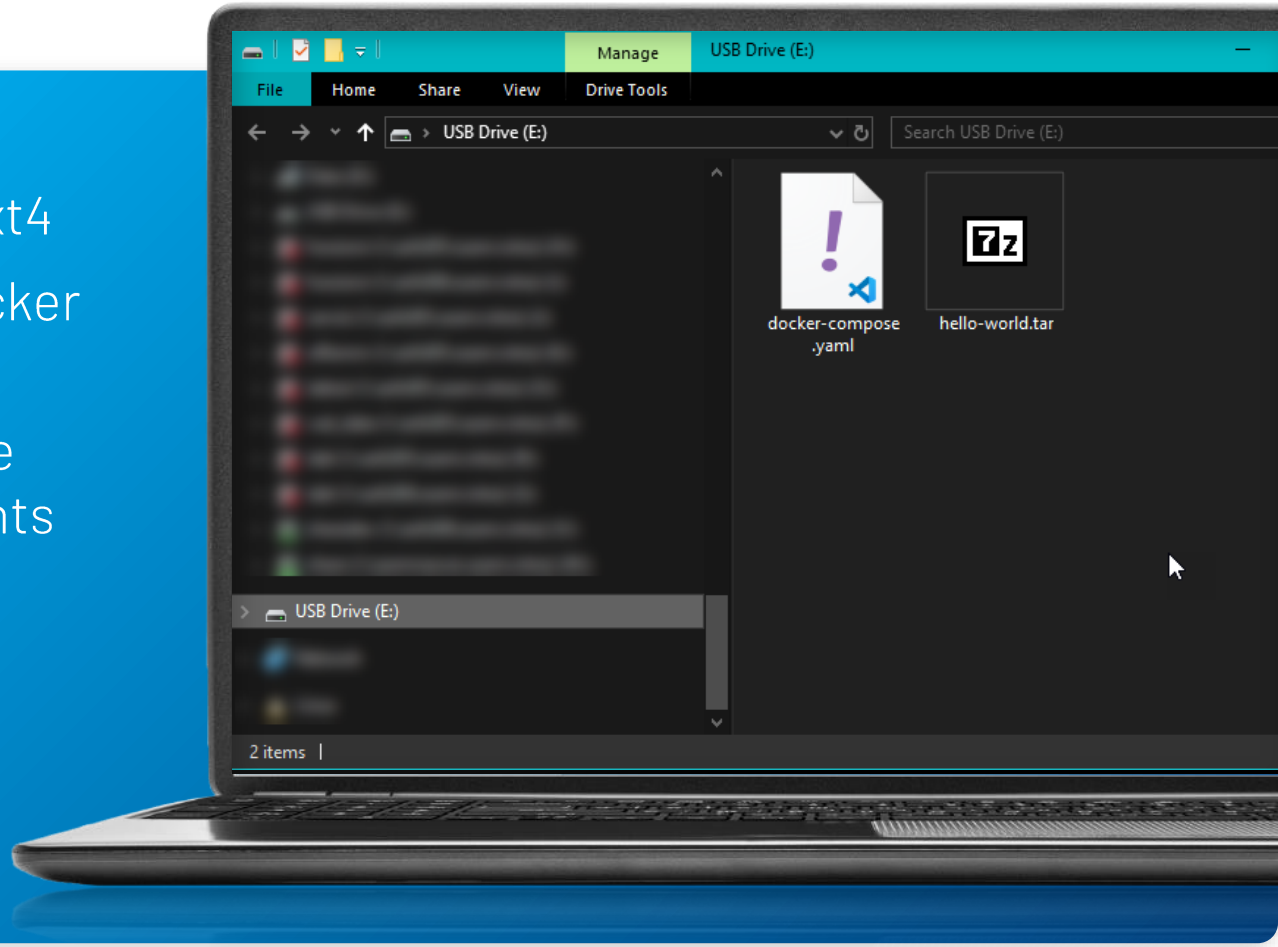
ASEM
A ROCKWELL AUTOMATION COMPANY

Loading containers from USB

Load a docker-compose.yaml and the relevant .tar image

- USB partition must be FAT32, exFAT or ext4
- The image «tar» file comes from the «docker save»
- A docker-compose.yaml file describes the container startup procedure and arguments

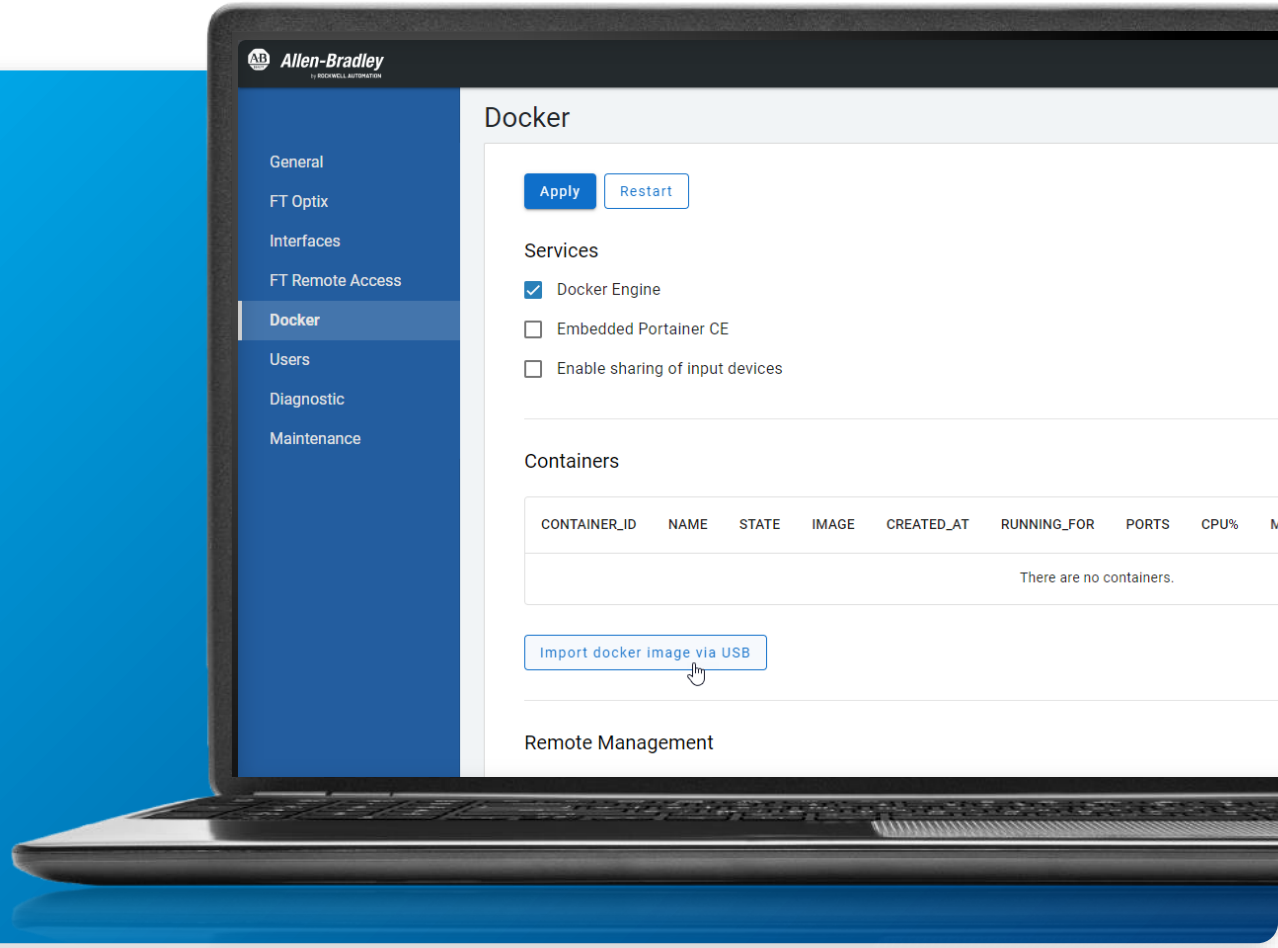
```
#file: docker-compose.yaml

services:
  hello-world:
    image: hello-world
```



Loading containers from USB

Load a docker-compose.yaml and the relevant .tar image

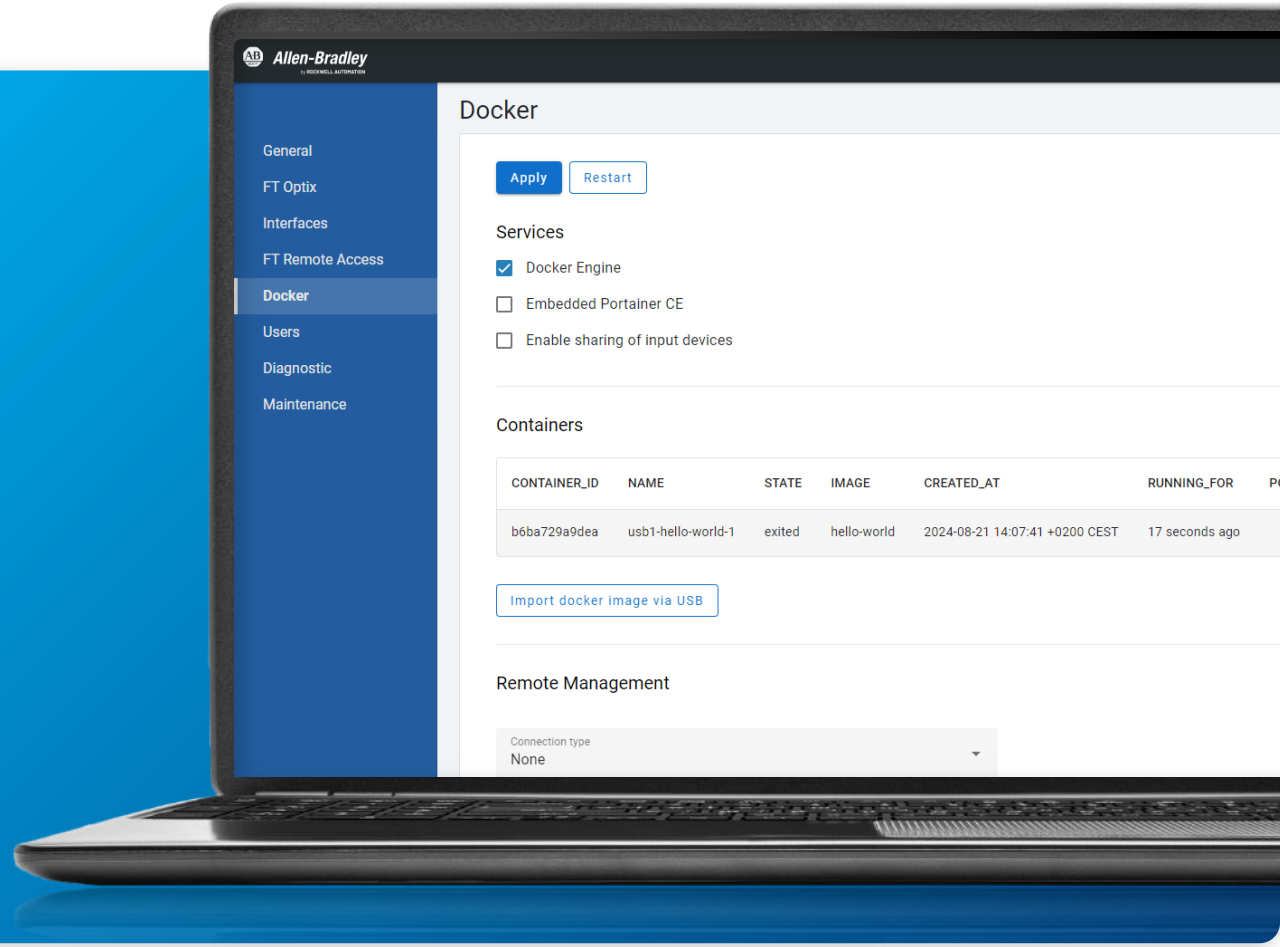
- Plug the USB key
- Import the container



Loading containers from USB

Load a docker-compose.yaml and the relevant .tar image

- Check the status



Example of nginx server

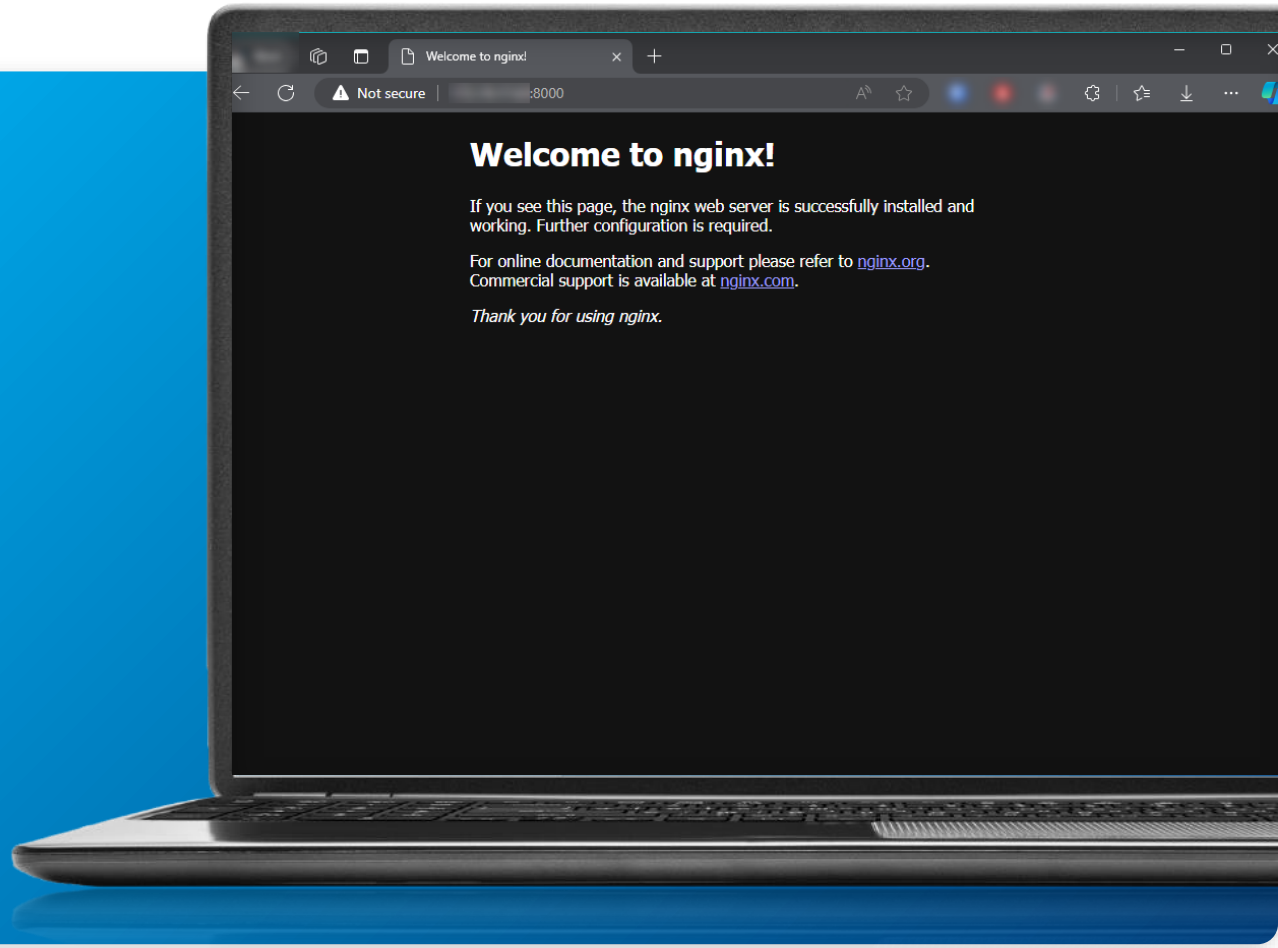
Services can be exposed by forwarding a TCP port

- Pull the image

```
$ docker pull --platform linux/arm64 nginx
$ docker save -o nginx.tar nginx
```

- Create the «docker-compose.yaml»
  - Network is set to «NAT» by default
  - Public address is the same as the Optix Edge module
  - Ports are forwarded in the stack file

```
services:
  web-server:
    image: nginx
    ports:
      - "8000:80"
```



Controlling Docker with Portainer

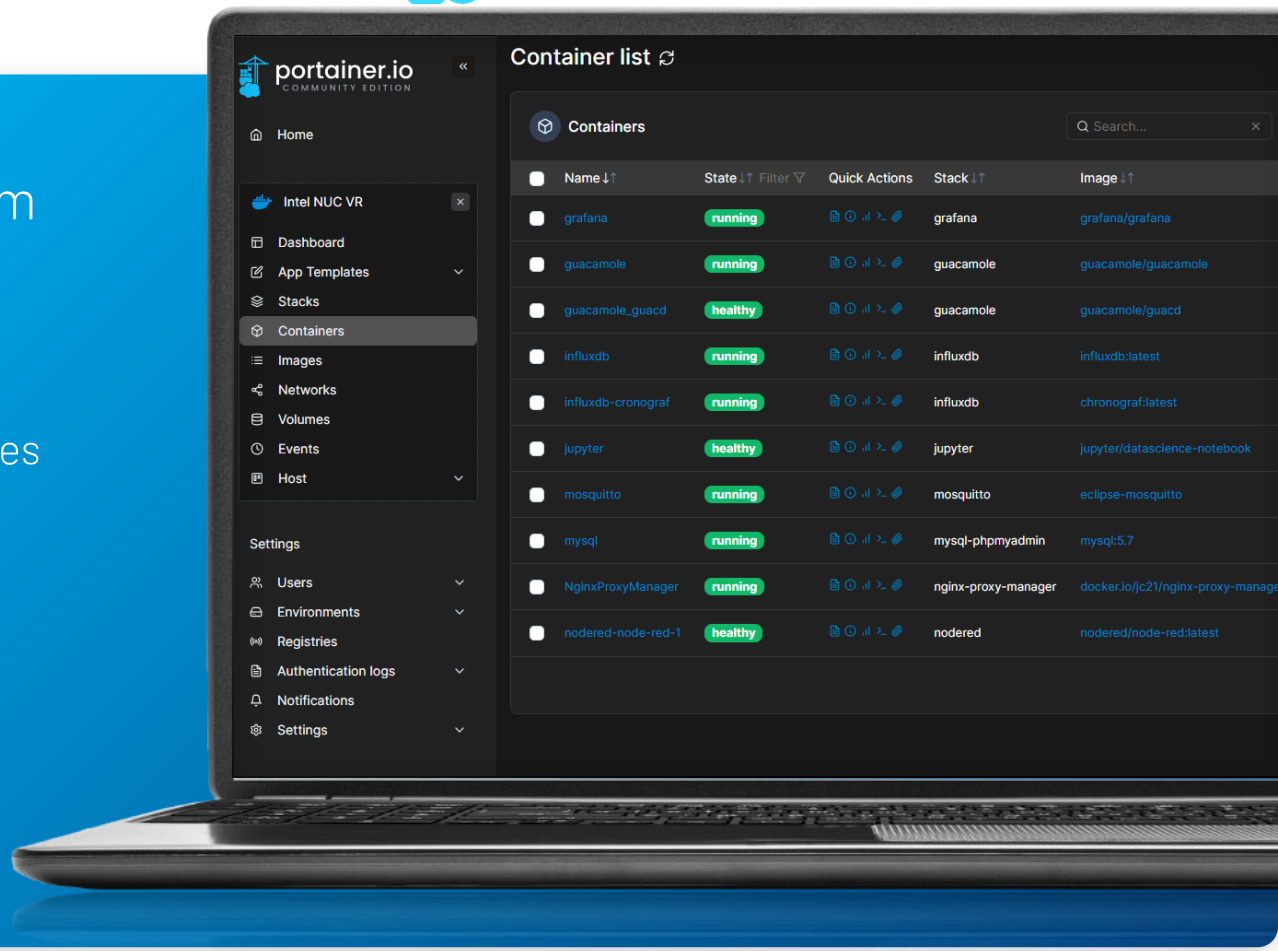
A user-friendly approach to containers

Portainer

Portainer is actually a Docker container

portainer.io

- Portainer is not a containerization platform
- Portainer is a GUI for Docker
- Comes in two flavours:
  - Portainer-ce which is free with some minor features limitations
  - Portainer-business which is paid and includes all features and support
- Allows connecting to multiple agents
- Supports both Dockerfile and Docker compose

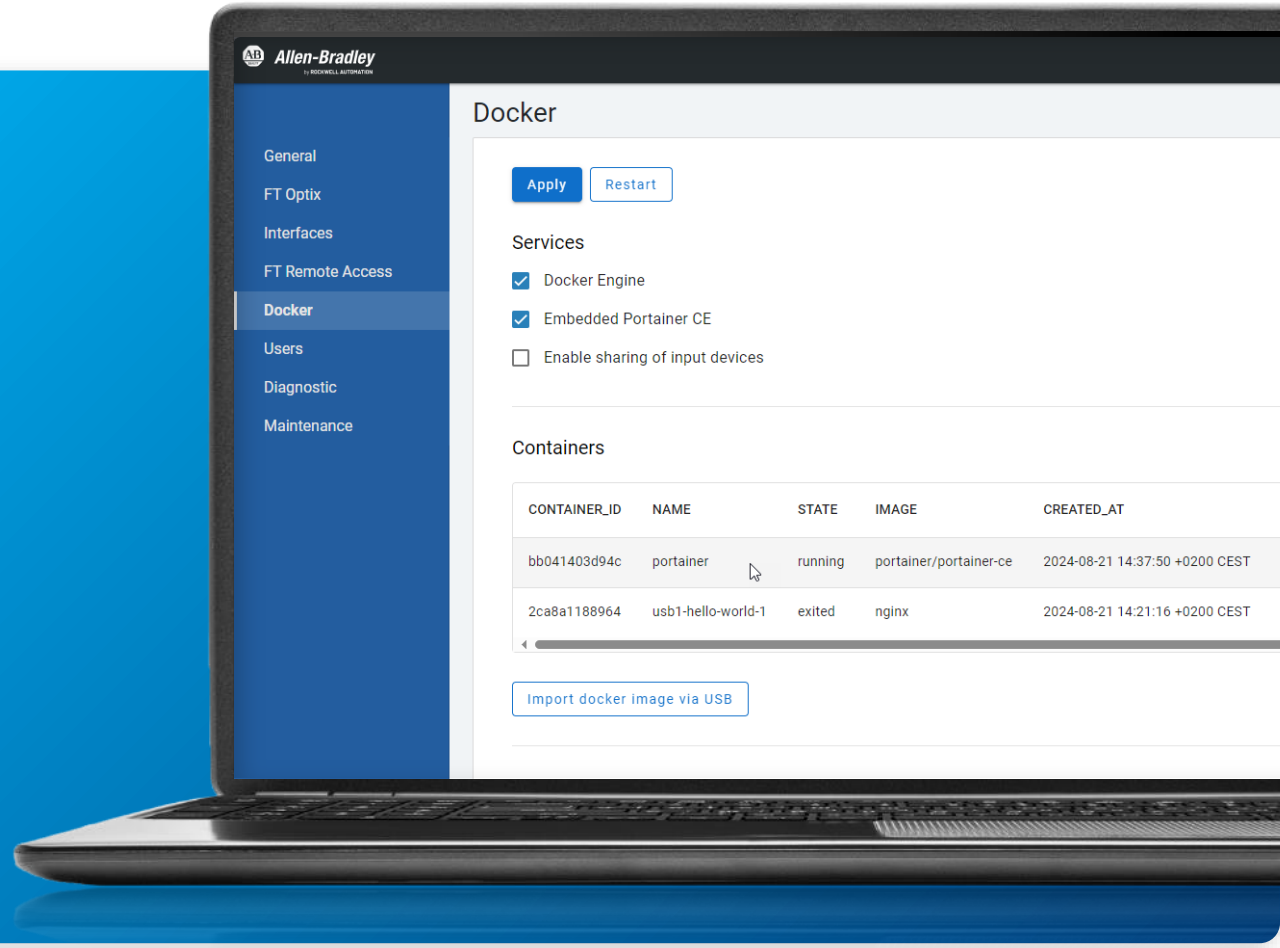


ASEM
A ROCKWELL AUTOMATION COMPANY

Enable Portainer from SystemManager

Tick the option and access the page

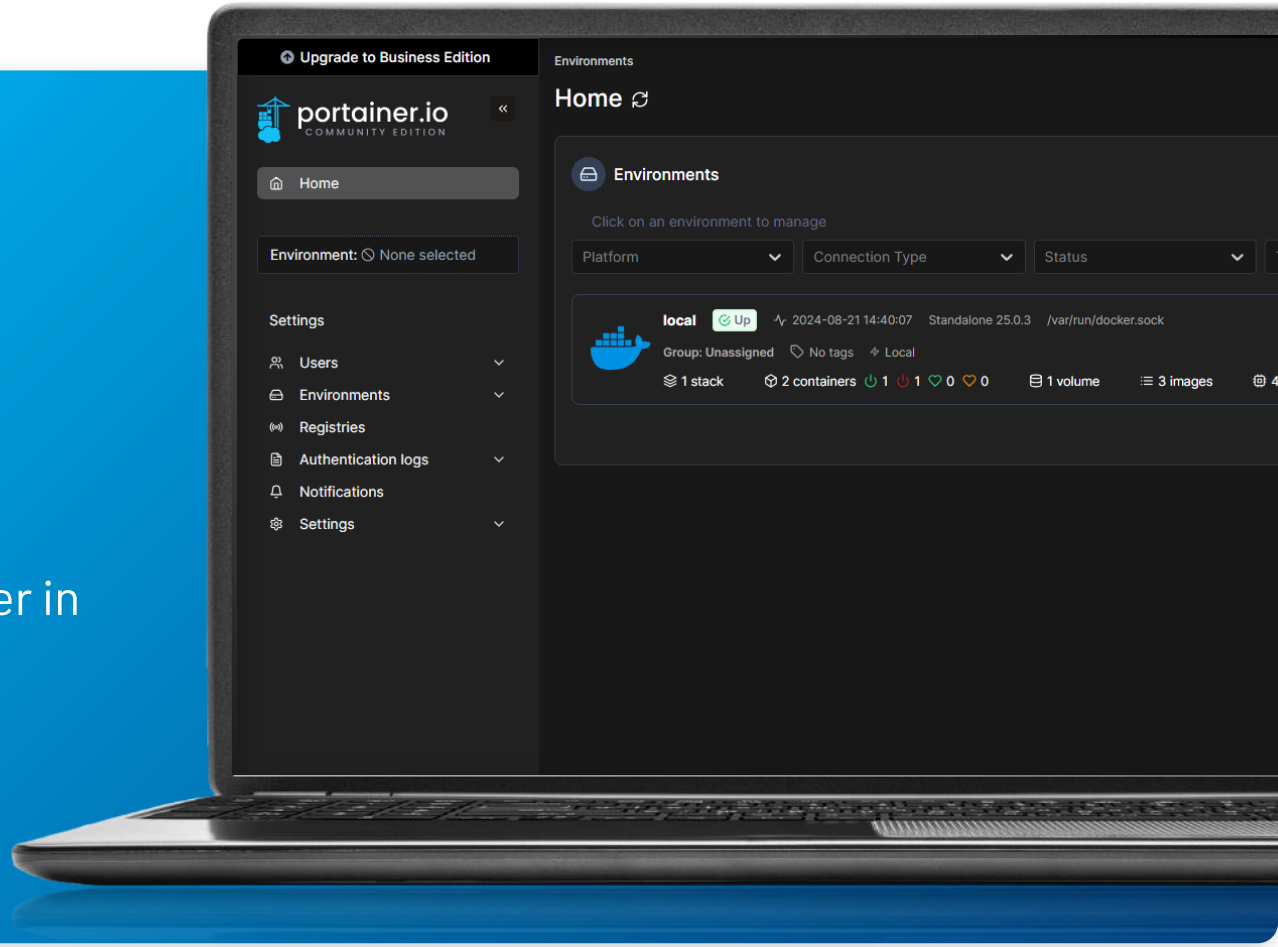
- Enable Portainer CE
- Reboot the device



Enable Portainer from SystemManager

Tick the option and access the page

- Access Portainer interface
  - **`https://ipaddress:9443`**
- Trust the self-signed certificate
- Configure admin password
  - Password complexity can be reduced later in user settings
- The «local» environment comes preconfigured



Starting containers with Portainer

Multiple ways to achieve the same result

Adding new container instance

- Each parameter must be configured manually
  - A little longer when configuring networks
  - Create the persistent volume before (if needed)
- Containers (and settings) are not saved in the backup file
  - Persistent volumes are not backed up[1]

Adding new container with a «stack»

- All parameters are loaded from the stack file
  - Stack file is actually a docker-compose
  - All container settings are loaded in a single shot
- Stacks are saved in the Portainer backup file
  - Easier to restore normal functioning
  - Persistent volumes are not backed up[1]

1. Some containers are available to backup Docker volumes and store them

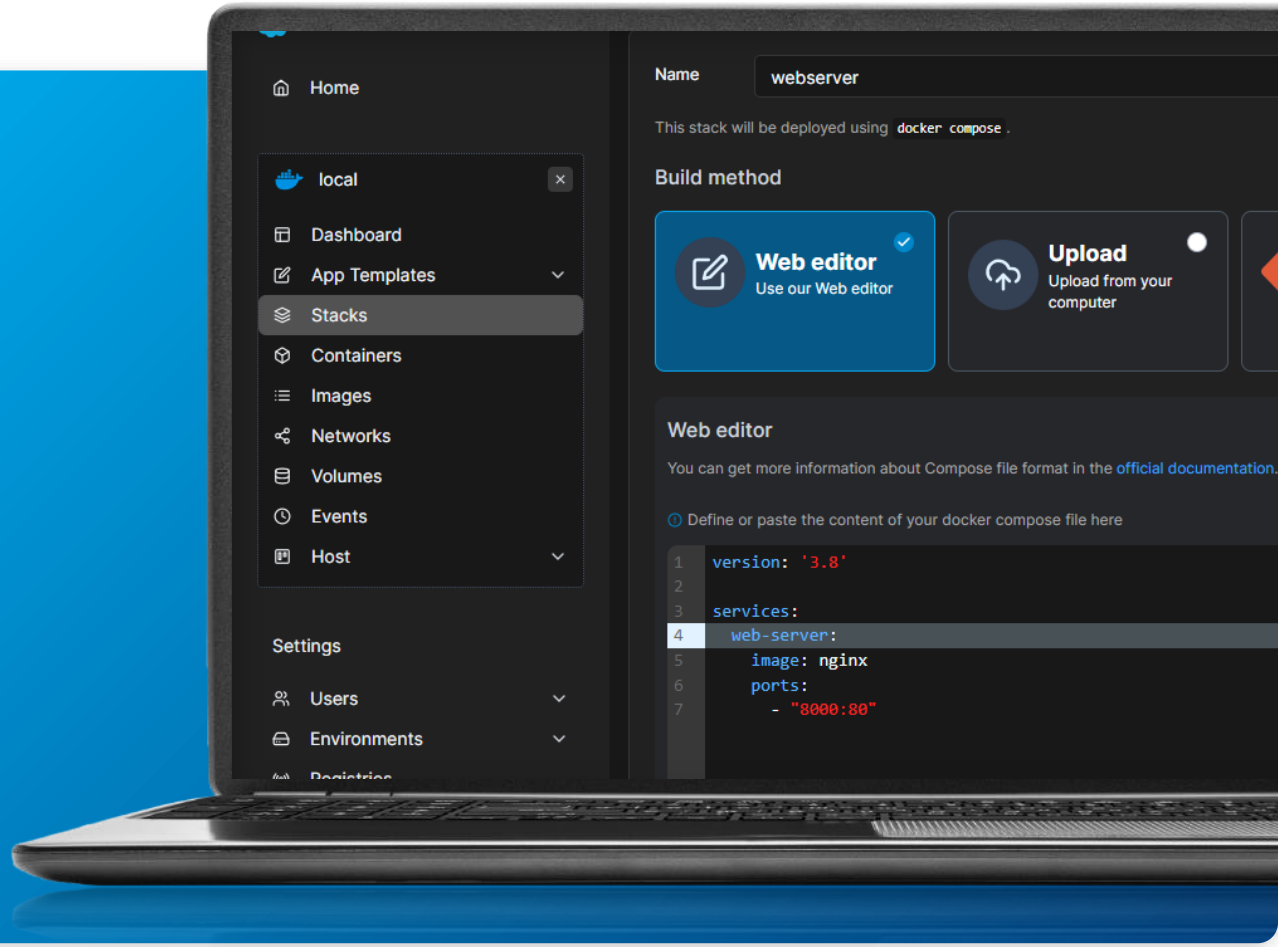
Loading a Docker container with Portainer

Look mum, no USB!

Creating a new stack

Stacks are just Docker compose files

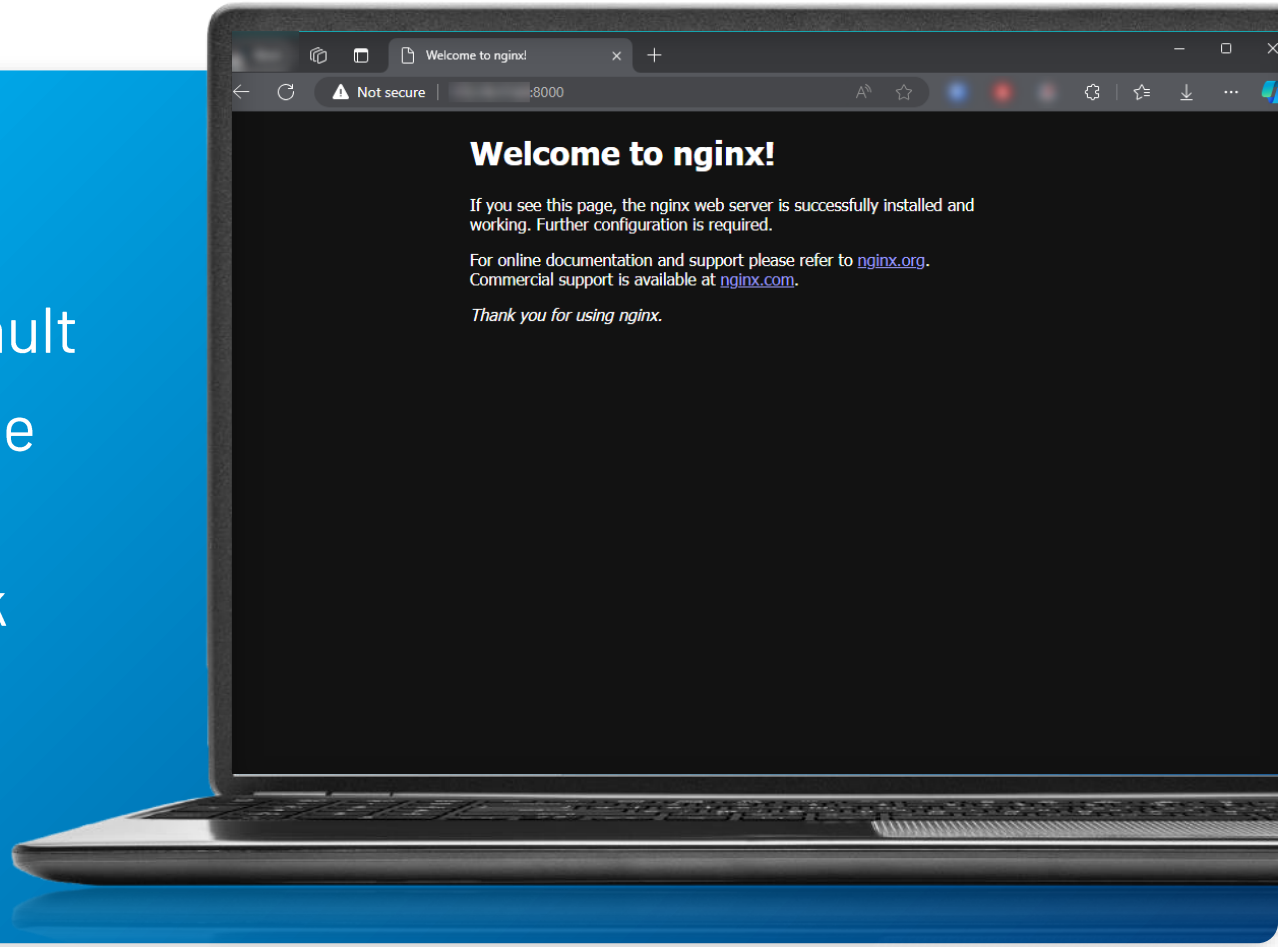
- Select the «local» environment
- Move to the «stacks» section
- Click «Add stack» in the up-right corner
  - Provide a stack name
  - Write the stack details
- Scroll down and deploy the stack



Example of nginx server

Same as before but started with Portainer

- Open the web server page
  - Network is set to «NAT» by default
  - Public address is the same as the Optix Edge module
  - Ports are forwarded in the stack file



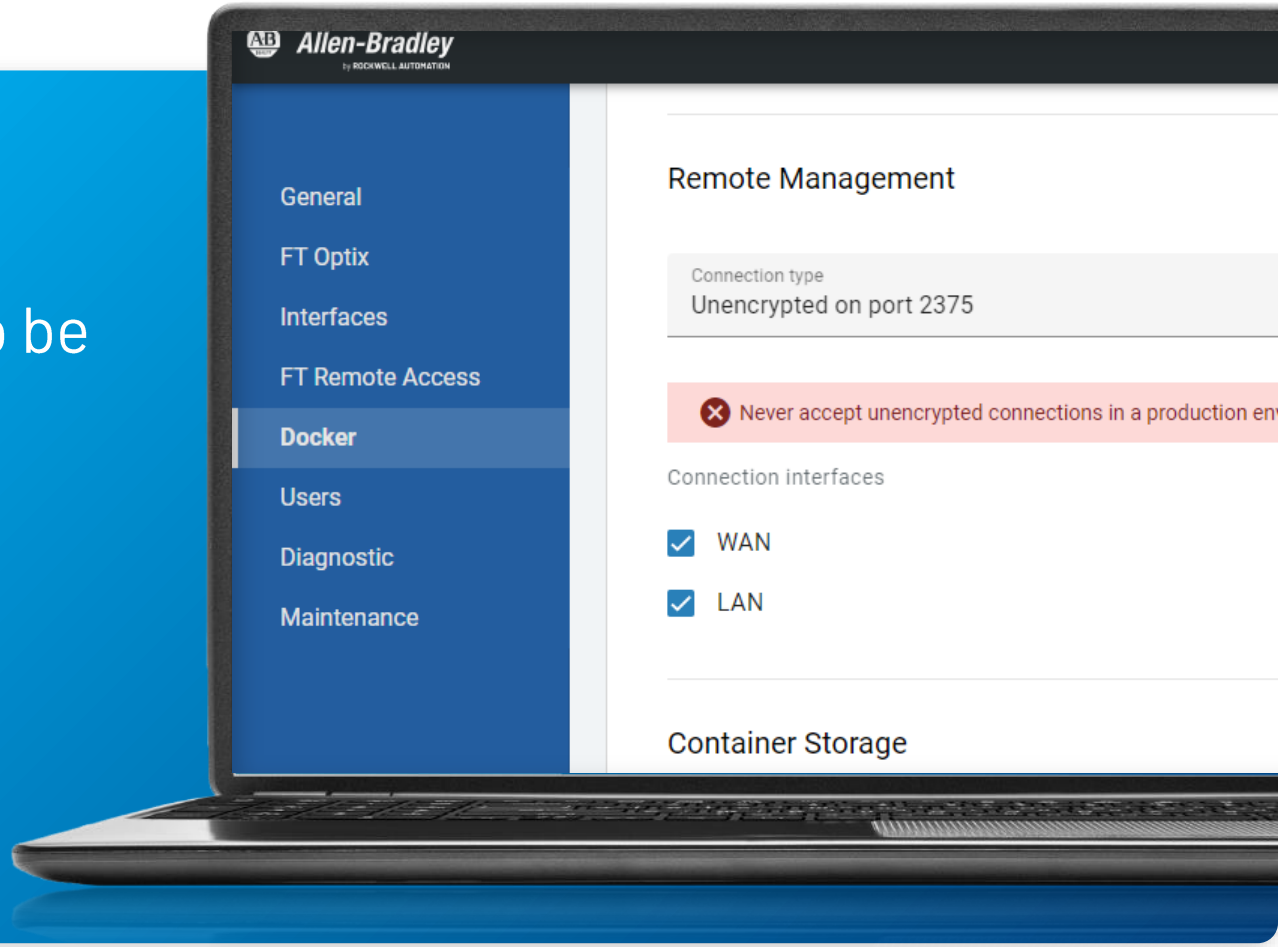
Docker CLI

Please, don't hack me

Enable Docker CLI SystemManager

Docker can connect to third-parties agents

- Enable Remote Management
  - Select which connection type to be used
  - Select which interfaces can be used to access the CLI

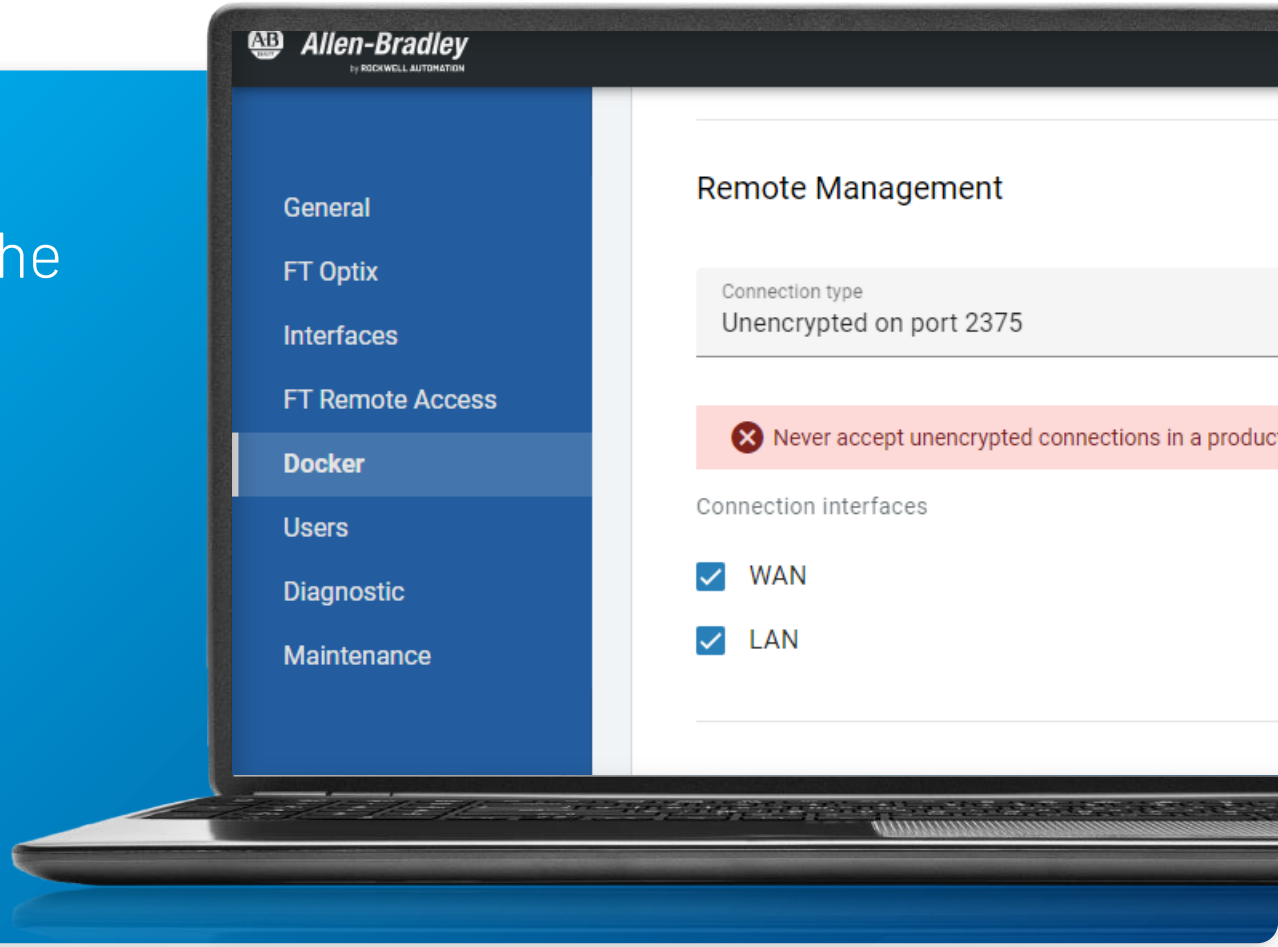


Connect the docker daemon to the agent

Docker can connect to third-parties agents

- Set the environment variable on the local machine
  - Unset the variable to restore normal functioning

```
$ export DOCKER_HOST=ipaddress:2375
$ […]
$ unset DOCKER_HOST
```



Sending commands to unencrypted agent

Works just like a local docker daemon

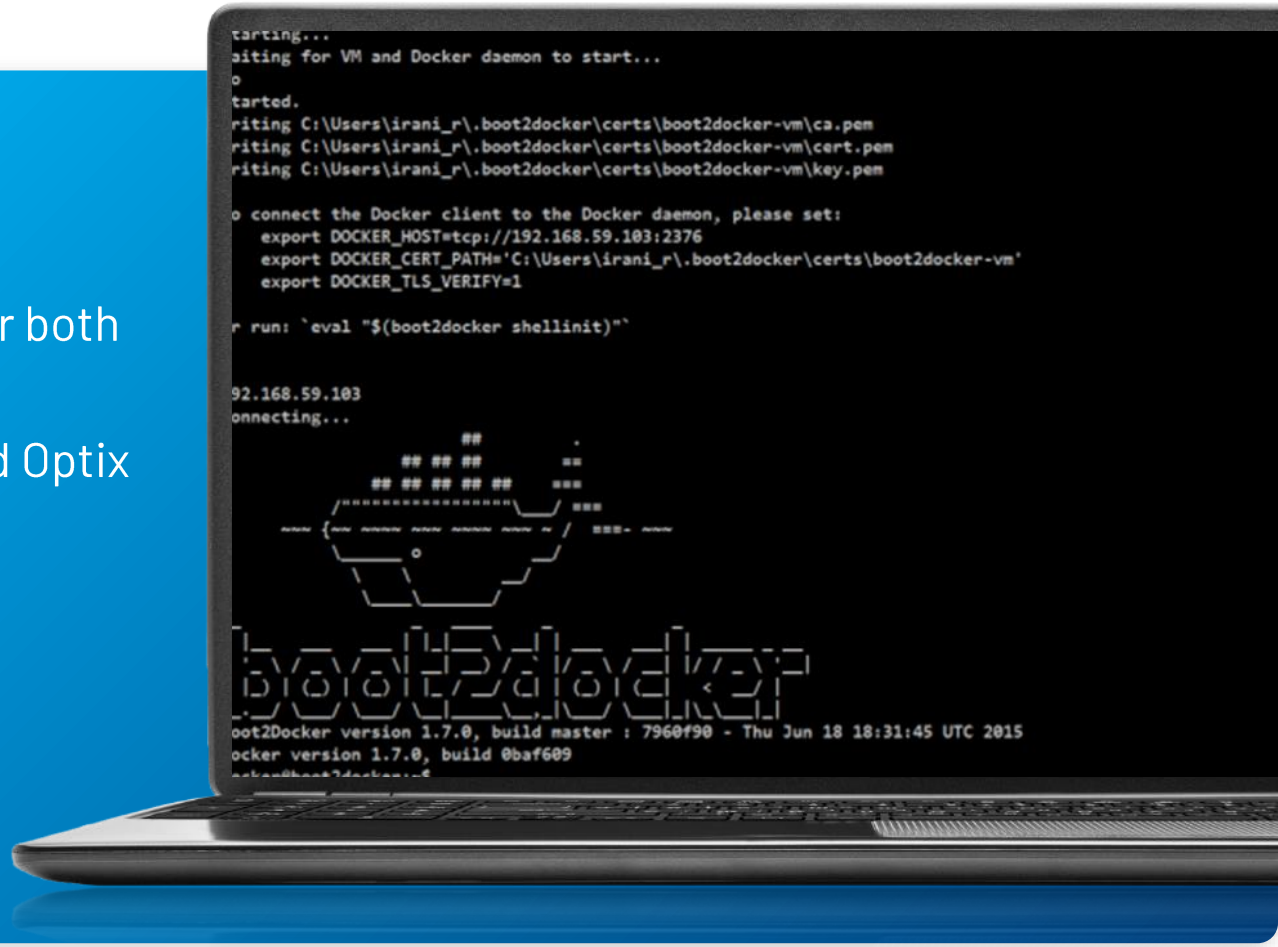
- Set the environment variable
- Send some commands to the agent
  - Works just like a local docker machine
  - Commands are forwarded to the Optix Edge automatically

```
$ export DOCKER_HOST=‘ipaddress:2375’ # Example: “192.168.0.1:2375”
$ docker ps –a # List the running containers
$ docker run --name web-server -d -p 8000:80 nginx # Starts nginx
$ unset DOCKER_HOST # Disconnect from module
```

Sending commands to encrypted agent

Works just like a local docker daemon

- Generate (or get) a set of certificates
  - Needs CA, public key and private key
    - CA and server certificates are the same for both client and Optix Edge
    - Private key files are different for client and Optix Edge (asymmetric encryption)
  - All files must be «.pem» format
- Set Remote Management to «Encrypted»
  - Load certificate files
  - Select which interfaces to listen



Sending commands to encrypted agent

Works just like a local docker daemon

Full commands sequence:

```
$ export DOCKER_HOST='ipaddress:2376’
$ export DOCKER_CERT_PATH=/path/to/client/certificates
$ export DOCKER_TLS_VERIFY=1
$ docker ps -a # List the running containers
$ docker run --name web-server -d -p 8000:80 nginx # Starts nginx
$ unset DOCKER_HOST # Disconnect from module
```

Access USB devices

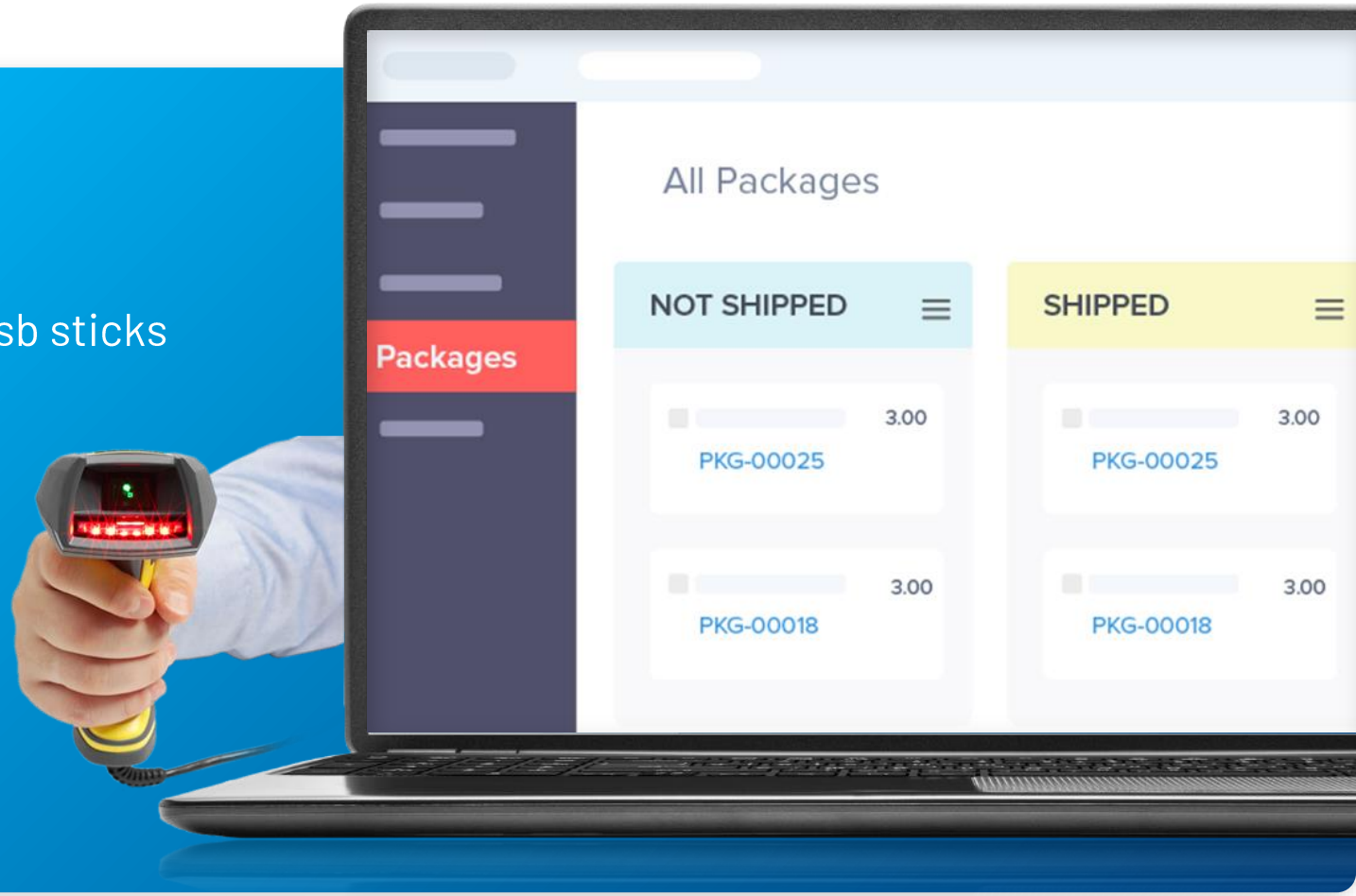
Tap, tap, tap...

Enabling Docker access to USB

Containers can access devices on the USB port

- Enable the feature in SystemManager
  - Can be used to access disk drives, usb sticks and more
- No need to add the device to the run command

```
$ export DOCKER_HOST=ipaddress:2375
$ docker run -it ubuntu bash
Ubuntu# apt update && apt install -y usbutils
Ubuntu# lsusb
1: Bus 001 Device 002: ID 23a9:ef18 USB DISK
Ubuntu# exit
$ unset DOCKER_HOST
```



Building a custom image

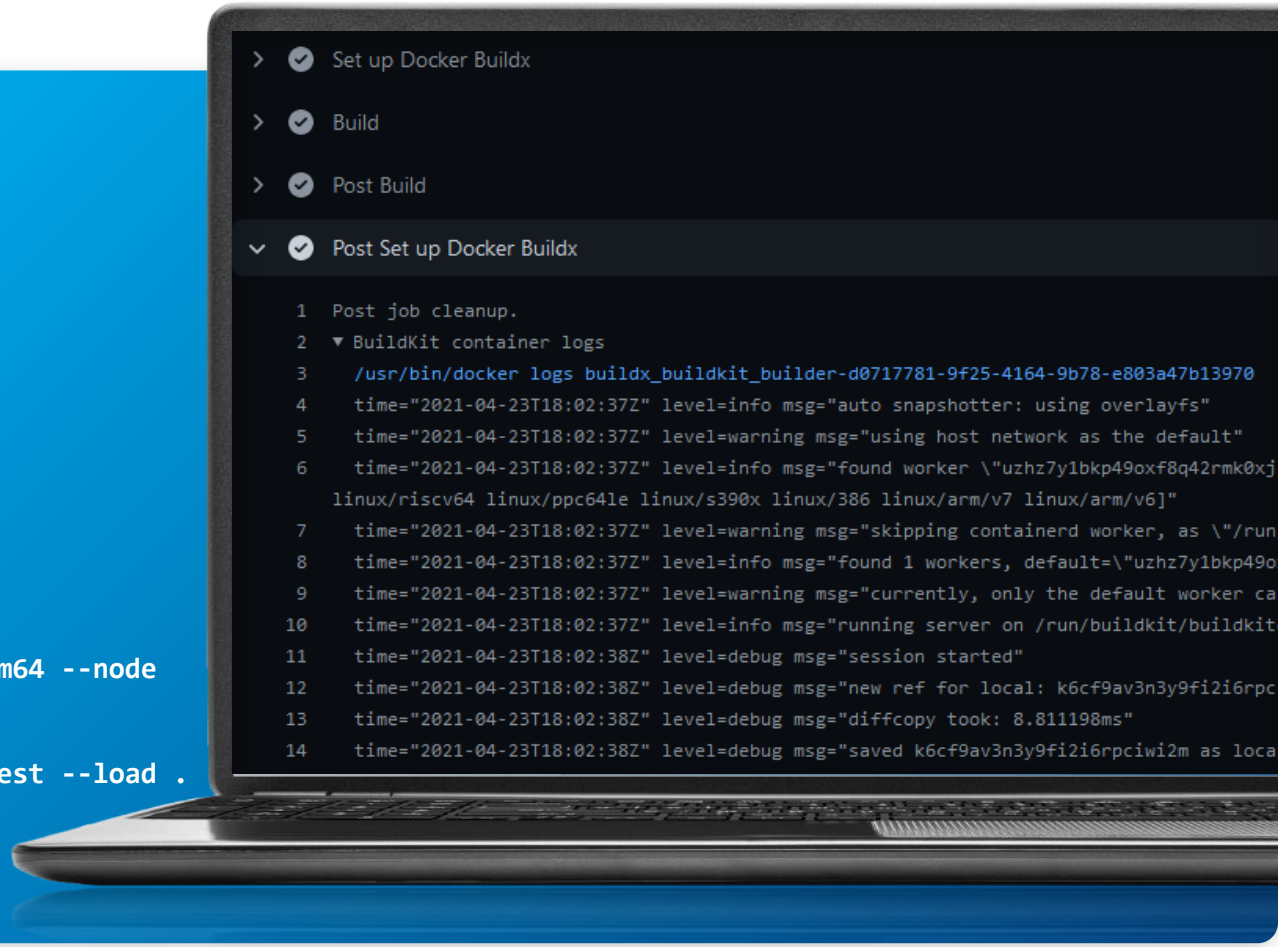
Same as on a PC, but need to specify the target architecture

Building a custom image

Make sure to cross compile with buildx

- Install buildx and dependencies
- Use buildx to create the image
- Export the tar file
  - Output file can be imported via USB, Docker CLI or Portainer

```
$ sudo apt-get install -y qemu-user-static
$ sudo docker buildx create --name optix-edge --platform linux/arm64 --node
node_optix-edge --driver docker-container --bootstrap -use
$ sudo docker buildx use optix-edge
$ sudo docker buildx build --platform linux/arm64 -t optix-edge_test --load .
$ sudo docker save -o optix-edge_test.tar optix-edge_test
```



ASEM
A ROCKWELL AUTOMATION COMPANY

FactoryTalk
by ROCKWELL AUTOMATION

Thank you

www.rockwellautomation.com